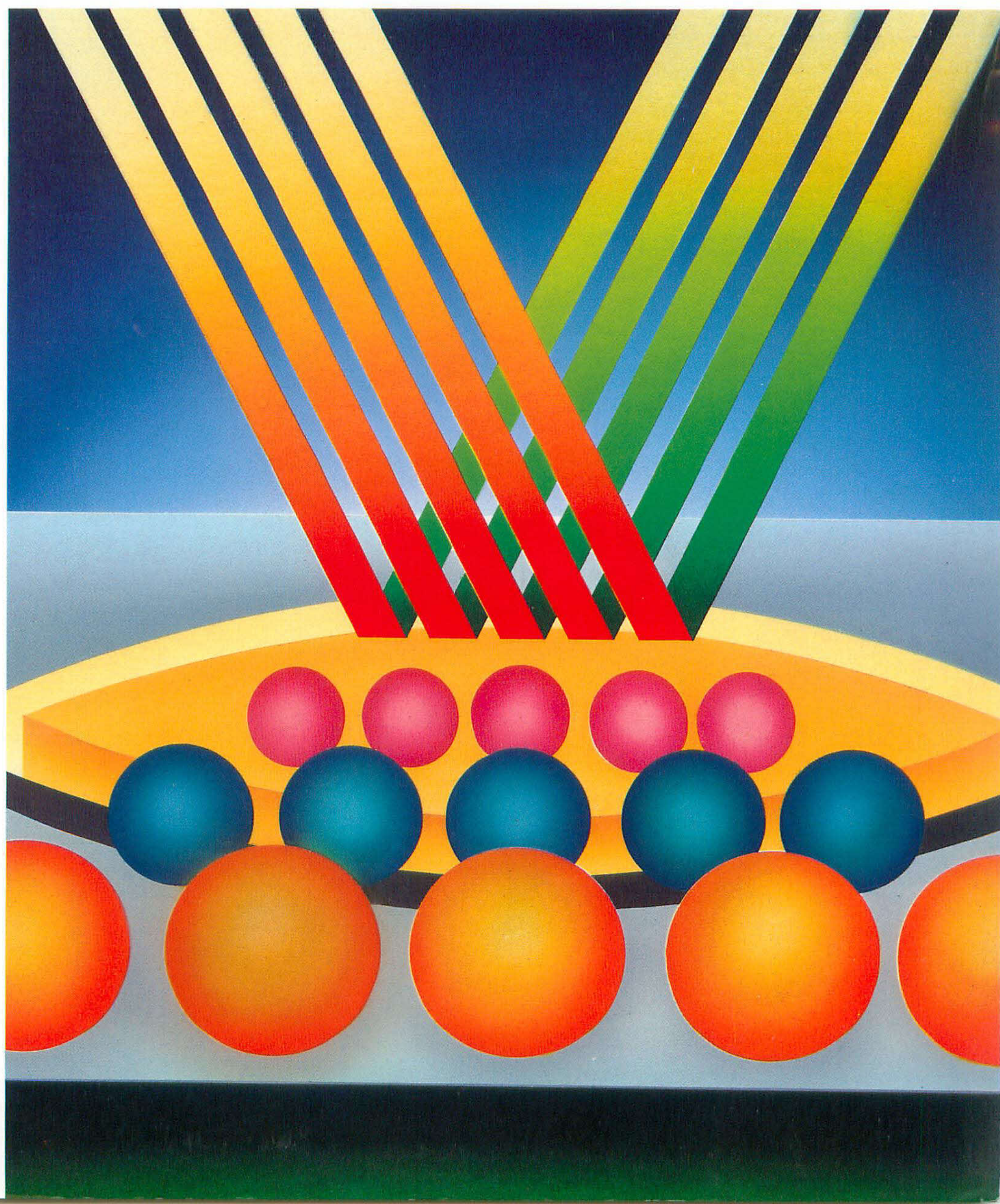
SPECIAL

トランジスタ技術

SPECIAL No.27

特集 ハードディスクとSCSI 活用技術のすべて

本格活用のためのハード&ソフトのすべてを詳解



トランジスタ技術 SPECIAL No.27

CONTENTS

特集 ハードディスクとSCSI活用技術のすべて

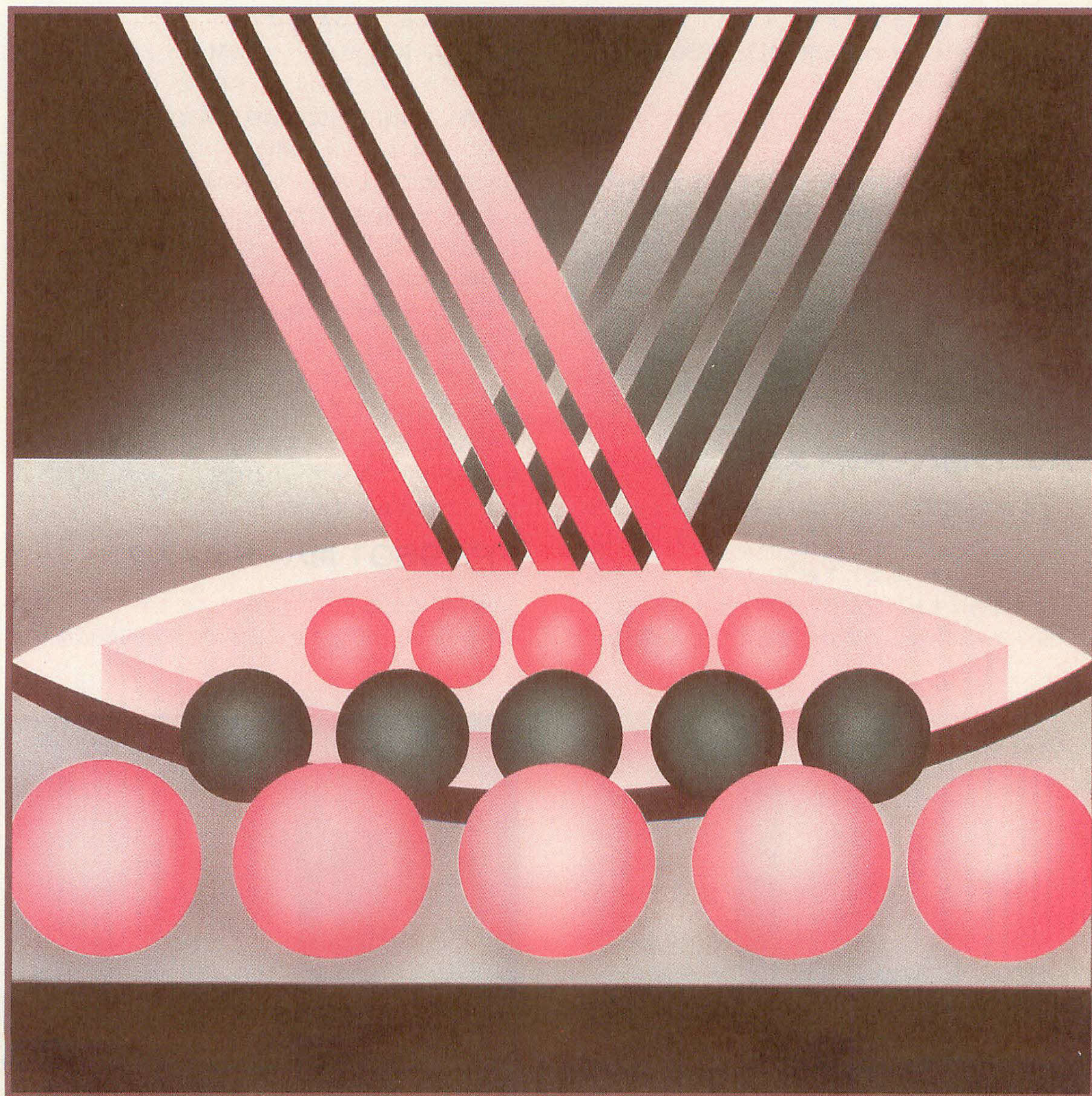
本格活用のためのハード&ソフトのすべてを詳解

表紙デザイン／簑原圭介

特集 ハードディスクとSCSI 活用技術のすべて

本格活用のためのハード&ソフトのすべてを詳解

ハードディスクはいつの間にかパソコンの必需品になりました．パーソナル仕様 HDD の密度と速度はとどまることなく向上し，値段と体積がみるみるうちに下降しています．そんな現在，ハードディスク関連の技術資料は皆無に等しく，内部についても見てくれと同じくブラックボックス化しています．今回の特集では全く中身について知られていない HDD のしくみと HDD とは切り離せない関係にある SCSI をあわせて解説します．



第0章

ハードディスクの歩み

容量，スピード，価格，体積，性能への挑戦

松村清明

1975年頃，14インチのハードディスクが付いたミニコンピュータが5千万円くらいしました．

このハードディスクの容量は5Mバイトだったと記憶しています．入出力端末はテレタイプでした．

5Mバイトでは大きなファイル・システムをのせることができません．そのため限られた実行形式ファイルとコンパイルのためのテンポラリ・ファイルのみがディスク上に置かれ，ソース・プログラムは紙テープで保存していました．

パソコンに600Mバイトのハードディスクが接続され，40Mバイトのハードディスクがディスカウント・ショップで山と積まれて売られている現在とは隔世の感があります．

もちろん，このような変化が急に起こったわけではありません．パソコンの発展とともにハードディスクも多くの小さなステップの積み重ねで発展してきたわけです．

●パソコン用ハードディスク

8インチ250Kバイトのフロッピ・ディスクとインテルのCPU8080AがCP/Mのもとでディスク・オペレーティング・システムとして動作したとき，システムとしての能力はそれ以前の紙テープ・ベースのミニコンピュータ・システムの能力を超えてしまいました．これは8080AとCP/Mという共通プラット・フォームの上に積み上げられたソフトウェアの重みと同時に，ディスク・オペレーティング・システムを簡単に構築することを可能にした8インチ・フロッピ・ディスク・ドライブによるところが大です．

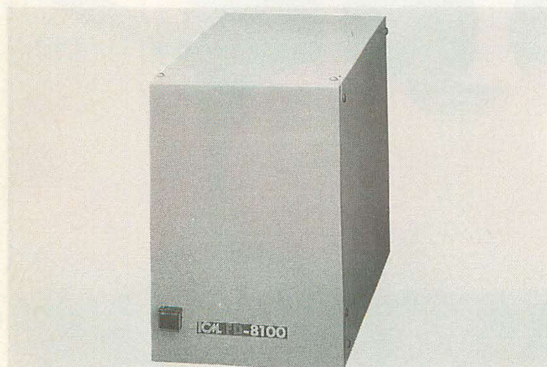
1980年，ICMが8インチ20Mバイトのハードディスク・ドライブを製品化し，NECのパソコンPC8001に接続して100万円位の価格で販売開始しました(写真1)．これはパソコン用のハードディスクとしては日本で初めてのものでした．フロッピ・ディスクと比べるとかなり高速でしたが，ホストCPU(Z80A)の能力が低いので，アプリケーション上で使用しても10倍の高速化というわけにはいきませんでした．

しかし，当時すでに発売されていた8086にはソフトウェアがなく，ホストCPUと周辺機器のアンバランスが目立った時代でした．

この製品はミニコンピュータ用のディスクと比べると圧倒的に安かったのですが，CPUの能力も合わせたトータルなスループットが低く，一般的に使用されるまでにはなりませんでした．

●PC9801用ハードディスク

1982年になるとST412タイプの5.25インチ10Mバイト・ハードディスクが日本でも入手可能になりま



〈写真1〉初期のパソコン用ハードディスク



〈写真2〉1トラック分のバッファを内蔵したハードディスク

した．ドライブ上には8ビット・シングルチップCPUが搭載されており，これがドライブのシークをコントロールしていました．このドライブにSASIインターフェースのディスク・コントローラを接続し，SASIバスでパソコンと接続しました．価格が50万円を切ったのもこの頃です．

また初代PC9801が発売され，ハードディスクを接続するパソコンも急速にPC8801からPC9801へと移行していきました．パソコン用のハードディスクでは先行していたサードパーティの物に加えて，NECもPC9801用ハードディスクを発売しました．

また，新機種PC9801EやPC9801Fではハードディスクからもブート・アップできるようにBIOSが変更されたことで，PC9801は名実ともにハードディスク対応のパソコンとなりました．

●ハードディスクの高速化

1984年頃にはPC9801のスピードも高速化され，周辺機器の高速化がパソコンのトータル・スループットを考えた上で重要になってきました．

そこで，インターリーブを使って5〜6回転で1トラックのデータをアクセスしていたそれまでの方法に対し，1トラック分のバッファをハードディスク・コントローラに持たせることで，高速リード/ライトを可能にしたハードディスク・ユニットも市場に登場しました(**写真2**)．

1986年頃になるとパソコンのメイン・メモリも640Kバイト・フル実装が一般化し，数100Kバイトのプログラム・サイズが一般化しました．そこで，プログラム起動時間がパソコンの操作感を左右するものとして重要になってきました．

プログラム起動時間，すなわちプログラム・ロードのスピードを決定するのはディスクのシーケンシャル・アクセス・スピードです．そこで，2トラック分のバッファを用意し，これをキャッシュ・バッファとして，ホスト側のデータ転送レートに影響されずにつぎのトラック分のデータをリード，もしくは前のトラックをライトできるようにして高速シーケンシャル・リード/ライトを可能にするといった方法も使われました(**写真3**)．

●普及型ハードディスク

そのうちハードディスクの価格はどんどん低下し，1987年には実勢価格で10万円以下のハードディスクが登場しました(**写真4**)．

この頃になるとこの価格のディスクを作るためにコントローラとディスク・ドライブを一体化し，性能的にはそれまでのハードディスクと同じものを3.5インチ・ディスクのサイズに押し込んでいました．

この頃のハードディスク・コントローラはすでにASICの塊になっていましたが，さらに複数のASICを複合化し，コンパクト化しました．

このように開発に多くの労力を費やしたものの，価格的なインパクトは大きく，市場規模は予想以上に拡大しました．

●操作環境

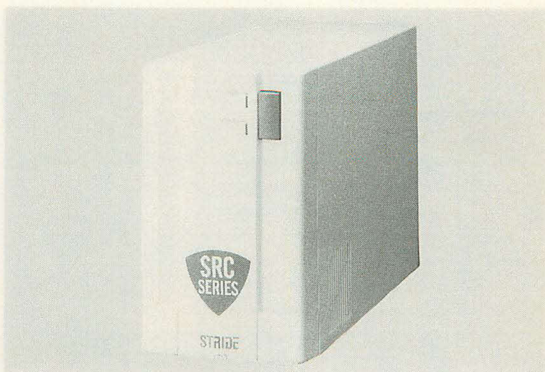
市場規模の拡大と共にハードディスクのユーザもパソコンをコンピュータとして使用する人達から，ブラックボックス化されたターン・キー・システムとして使用する人達へ広がってきました．

こういった人達にとってすればハードディスクもソフトウェアを使うための単なる手段です．したがってソフトウェアのインストールさえできればとりあえずハードディスクの使い方を覚える必要はありません．

そこで，画面に出てくる指示にしたがって操作を行えば，目的のソフトウェアがインストールされ，そのソフトウェアがハードディスク上で使用可能になるような操作環境(**写真5**)があたりまえのこととして各社から提供されています．



〈写真3〉キャッシュ・バッファの内蔵ハードディスク



〈写真4〉普及型ハードディスク

この頃から，パソコン用ハードディスクというもののとらえ方がハードディスクという単なるハードウェアを提供するのではなく，ソフトウェアを快適に使用するための環境を提供するのだという方向に変わってきたわけです．

●インターフェースの高速化

コンピュータの世界では高速化の要求はとどまるところを知りません．

1987年には1M～16Mバイトのディスク・キャッシュを持ち，16ビットCPUであるV40でコントロールするハードディスク・コントローラを開発し，製品化しました．

しかし，ディスクをここまで高速化すると今まで表に現れていなかったことが問題になりました．PC9801のハードディスク・インターフェースのデータ転送レートがDMA転送速度のボトル・ネックで限界に達してしまったのでハードディスクをいくら高速化してもほとんどスピード・アップしないのです．

そこで，ディスク・インターフェースのデータ転送レートを上げる方法を考えないとせっかく作った製品も出荷できないということになりました．

こうともなると，さて困ったものです．PC9801ではDMAコントローラ(i8237)の関係で，シングル・トランスファ・モードDMAのみしか使えません．これではデータ・レートは最大速度で400Kバイト/sくらいのものです．これに対してキャッシュ・ヒットによるメリットを出すためにはせめて800Kバイト/sくらいの転送速度が必要です．そこで，Vシリーズや286以上のCPUで可能な入出力ストリング命令を用いてCPUにデータ転送させることにしました．この方法は少しずつ転送しても効果がありません．256バイト以上のような単位でステータス・チェックなしに転送できることが必要です．そこで，ゲートアレイとSRAMを用いてFIFOを構成し，このSRAMの中にバッファリングする転送方法を実現しました．

1988年には記録方式をMFM変調方式からRLL(2, 7)変調方式へ変更すると同時に，データ転送レートを上げた物へと移行しました．これにより，5Mbpsのデータ転送レートが10Mbpsになったことで同一ヘッド，シリンダ数で2倍の記録容量となり，20万円以下で80Mバイトというディスクを実現しました．

そして1990年にいたると2.5インチ40Mバイト・ハードディスクが登場します(**写真6**)．

●ハードディスクの将来

さて，ハードディスクはこれからどのようになるのでしょうか．他のメモリに置き換えられてしまうのでしょうか．

今までも幾度となく繰り返されてきた疑問ですが，ハードディスクはいまだに，同等もしくはより大きくなって二次記憶装置の主流をなしています．

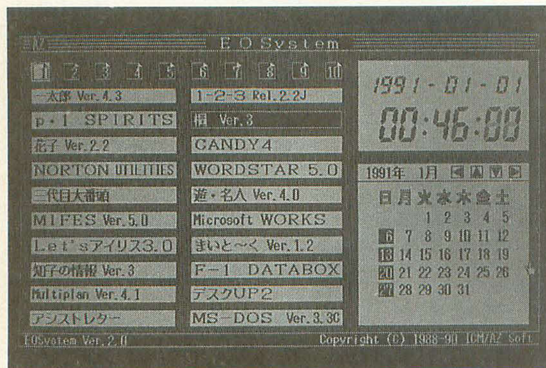
RAMもしくはEEPROMのような半導体メモリに変わるという話をよく聞きますが，半導体メモリに現状の価格低下曲線を大きく下まわるような急激な価格低下がない限り，この先5年間でハードディスクとビット・コストが逆転するようなことはなさそうです．

ただし，数Mバイト以下の容量では半導体メモリが使われるようになるでしょう．高記録密度化が著しいハードディスクでは数10Mバイト以下の容量のディスクを作っても安くならないからです．また，耐環境性要求のきつい分野(100Gをはるかに超えるような耐衝撃性，高高度や海底のように気圧が極端に低いまたは高い所での使用，低温もしくは高温での使用など)でもかなり半導体メモリが使われると思われます．

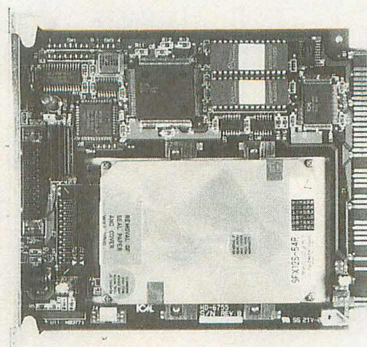
しかし，それ以外の分野ではやはりハードディスクは使い続けられていくでしょう．

●ハードディスクのサイズは

次にディスクのサイズはどうでしょうか．

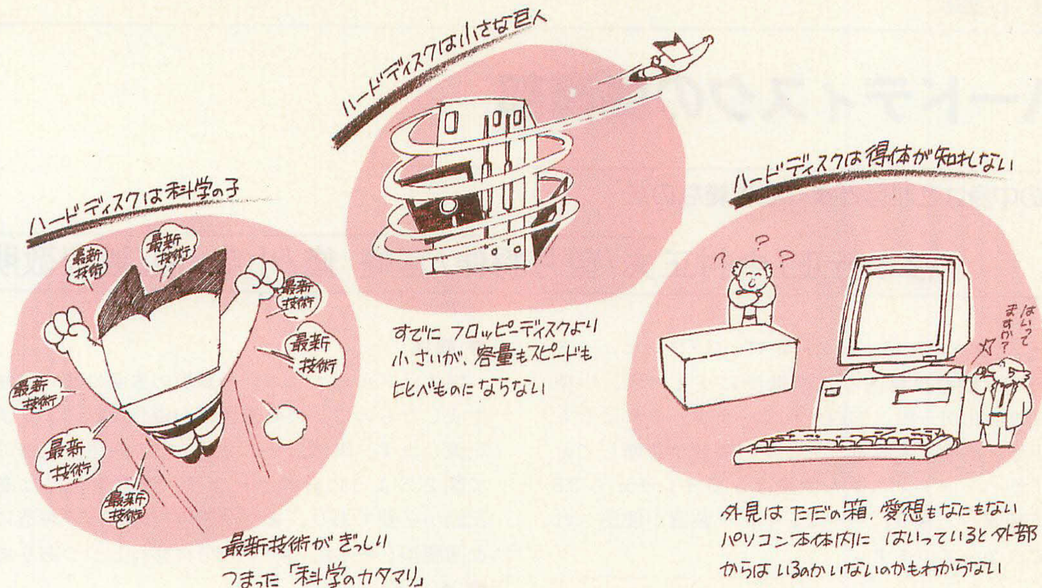


〈写真5〉操作環境を提供する(EOシステム)



〈写真6〉2.5インチのハードディスク(C FLAT)

〈ハードディスクはこんな奴だ〉



現在，14インチ，11もしくは10インチおよび8インチ・サイズの汎用機，高位ミニコン用の大型ディスクを除くと5.25インチ，3.5インチそれに2.5インチ・サイズのディスクが現在のミニコン，ワークステーション，パソコン用ディスクとして使用されています．

5.25インチ・ディスクは300Mバイト以上2Gバイト以下の容量が主流です．ミニコンやデスク・サイド型ワークステーションで主に使用されています．

3.5インチ・ディスクは40Mバイトから500Mバイトの容量帯を占め，小型ワークステーションやパソコンで使用されています．

これに対して2.5インチ・ディスクは20Mバイトから100Mバイトの容量帯で主にノートブック・パソコンで使用されています．2.5インチ・ディスクはまだこれからの商品ですが，1992年には40Mバイト以下の容量帯で3.5インチ・ディスクにとって変わると考えられます．このように今後も各サイズのディスクごとに互いに容量的に若干オーバ・ラップする関係を続けつつ，高容量化していくでしょう．

2.5インチ・ディスクのつぎはどうでしょうか．

今，各社が開発にかかっているのは1.89インチ(いわゆるICカード・サイズ)のようです．1991年中，遅くとも1992年にはサンプルが出荷されるでしょう．ただし，2.5インチ・ディスクのときのノートブック・パソコンのように絶対的な市場の要求があるわけではないので，製品が立ち上がるかどうかはマーケット次第というところでしょう．

1.89インチのつぎは1.5インチ以下といわれています．マッチ箱サイズ位です．このサイズとなると電子回路の実装方法が問題になります．通常のパッケージングされたICを並べると2～3個のLSIを載せるのがやっとの大きさだからです．チップ・オン・ボードやTABなどの複合化された実装方法が要求されると思われます．

●ハードディスクのこれから

CPUの処理能力は2000年までに1000MIPSを超えるといわれています．この頃のデスク・トップ・パソコンのメモリ構成は64K～256Kバイト程度の内蔵キャッシュから始まり，256K～1Mバイト程度の外部キャッシュ，64M～数Gバイトのメイン・メモリを持ち，数Gバイトのディスク・キャッシュ，もしくは半導体ディスクを通して数100G～数Tバイトのディスクを持つことになるでしょう．

マルチメディアから進んでバーチャル・リアリティの世界でマン・マシン・インターフェースを構成するためにはこれ位のハードウェアが必要であると同時にきわめてコンパクトに構成できるようになるでしょう．

このようなディスクを実現するためには，1980年から1990年までの10年間の変化にも増して急速に技術革新をしていかなければならず，増々多くの努力をしていかなければならないと思いを新たにするところです．

〈写真提供：ICM〉

第1章

ハードディスクの機構部

箱の中身はこれほどまでに複雑なのだ

杉本行正/中村正夫/望月将伸/宮崎 修/小寺 勉/渡辺敬明

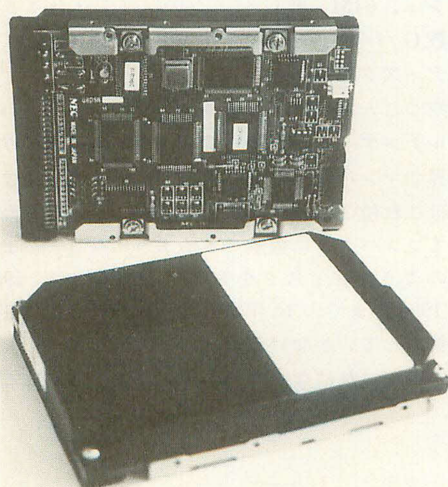
パーソナル・コンピュータ，オフィス・コンピュータなどのOA機器は急速な技術進歩に支えられ，小型化・高性能化が一段と加速されています．それにともない小型・高速・大容量ファイルの需要が急増しており，小型ハードディスクも主役は5.25インチから3.5インチ装置へと移り，今や2.5インチ装置が使用されるまでになっています．

機構部

ハードディスク・ドライブの構成は**写真1**に示すように，機構部とこれらを制御する電子回路部に大別されます．それらの機能ブロック図を**図1**に，代表的な製品(日本電気)を**表1**に示します．

● 機構部の構成

機構部は**写真2**に示すようにベース，カバーから構成されるDE(ディスク・エンクロージャ)内部に磁気ディスク媒体，磁気ヘッド，スピンドル機構，ヘッド・ポジショニング(ヘッド位置決め)機構，塵埃除去機構などが機能的に配置され構成されています．



〈写真1〉3.5インチ・ハードディスク装置
(上：電子回路部，下：機構部)〔D3855 日本電気(株)〕

● 媒体

媒体はアルミニウム合金基板の表面に薄い磁性膜を生成したもので，この磁性膜の磁化反転により情報を記憶します．媒体はその寸法，磁性膜生成工法によって**図2**のように分類されます．大容量化には高密度記録が必要であり，また高密度記録媒体の製造に有利な薄膜型(メッキ，スパッタ)に移行しつつあります．

● ヘッド

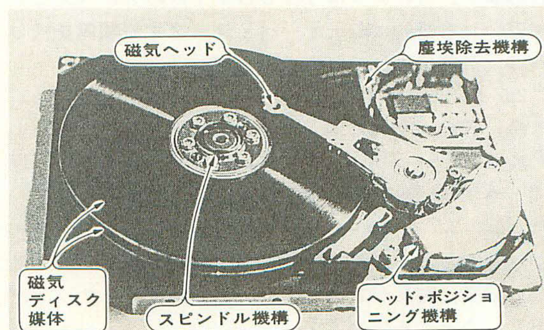
ヘッドはリード/ライト・トランスジューサを備えたスライダ部とそれを支持するサスペンション部から構成されます．媒体が回転することによる運動エネルギと空気粘性から，スライダ部は媒体表面上をわずかに(サブミクロン：0.1～0.2μm)浮上します．スライダ部を飛行機のボーイング747(約70m)にたとえるなら，高度3mmで飛行しながら尾翼付近に取り付けられたトランスジューサにより，磁化パターンを記録・再生していることに相当します．

媒体の高密度記録化に対応してトランスジューサ部の製法もフェライト→MIG(メタル・イン・ギャップ)→TFH(薄膜ヘッド)と記録・再生効率の向上が図られています．

● スピンドル機構

スピンドル機構は媒体を高精度に回転させる機構で，軸受け部とモータ部により構成されます．

軸受け部は回転精度に大きな影響を与えるため，1対の高精度ボール・ベアリングが使用されます．モー



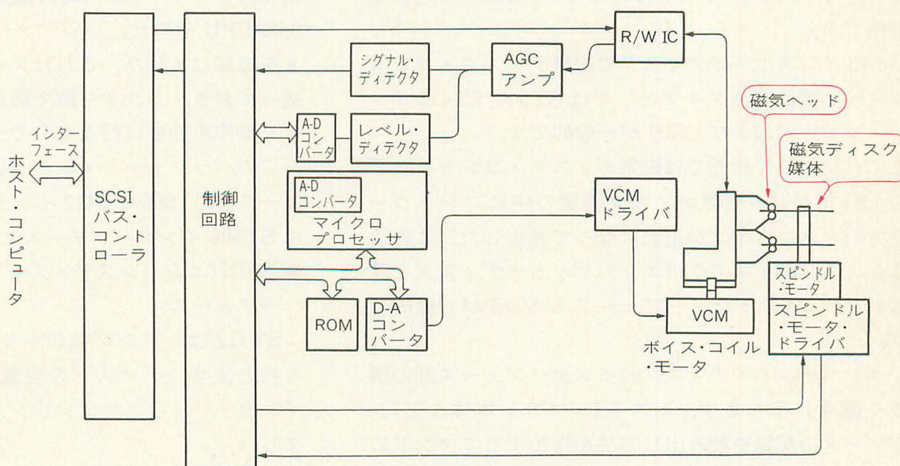
〈写真2〉ハードディスク・ドライブの内部

〈表 1〉
小型ハードディスク装置の仕様〔日本電気㈱〕

		3.5 インチ HDD				5.25 インチ HDD	
型名		D3142	D3×61*	D3×35*	D3×55/56*	D5×62*	D5×82*
記憶容量(M バイト)	アンフォーマット	53.4	134.45	56.4	131.3	385	765/763
	フォーマット	42.0	118/114	45.0	105.0	300/328	664/649
平均シーク時間(ms)		28	20	25	25/19	18	16
データ転送速度(K バイト/s)		625	1250	1500	1500	1250	1875
インターフェース		ST506	ESDI/SCSI/PC/AT	SCSI/PC/AT	SCSI/PC/AT	ESDI/SCSI	ESDI/SCSI
外形寸法(mm)	幅	101.6	101.6	101.6	101.6	146	146
	奥行き	146	146	146	146	208	208
	高さ	41.3	41.3	25.4	25.4	82	82
重量(kg)		1.0 以下	1.0 以下	0.5 以下	0.5 以下	3.5 以下	3.5 以下

*：×=1：ST インターフェース, ×=6：ESDI, ×=7：PC/AT, ×=8：SCSI

〈図 1〉
ハードディスク装置の機能ブロック図



タ部は軸受け部に直結された，長時間使用が可能なDC ブラシレス・モータが採用されています．

このモータには従来，回転位置検出にホール素子などが組み込まれていましたが，近年の小型化指向から検出素子のないセンサレス・タイプが採用され始めています．

● ヘッド・ポジショニング機構

これはヘッドを媒体上の目的位置(シリンダ)に位置決めするもので，回転自在のピボットに保持されたアームの一端にヘッドが取り付けられ，他端に駆動モードが取り付けられています．

従来の駆動モータはステッピング・モータ，DC トルク・モータなどが使用されていましたが，アクセス・タイムの短縮のため低慣性高トルク型のボイス・コイル・モータが採用されています．

また，そのボイス・コイル・モータの心臓部ともいうべき磁石には現在入手可能な最高性能の永久磁石であるNd-Fe-B(ネオジム・マグネット)系が使用されています．

● 塵埃除去機構

ヘッドの浮上量は前述したように非常に微小であり，目に見えない微細な塵埃でもヘッドと媒体の間に入り込むと媒体が損傷し，記録された磁化パターンが失われることになります．したがって DE 内部は高清浄度に保つ必要があり，万一の外部塵埃やわずかな内部発塵に備え，塵埃除去機構としてエア・フィルタが設けられています．

〈図 2〉 **寸法と製法による媒体の分類**

寸　法(内径×外径×板厚)
- 5.25 インチ(130×40×1.905mm)
- 3.5 インチ（ 95×25×1.27mm)
- 2.5 インチ（ 65×20×0.89mm)

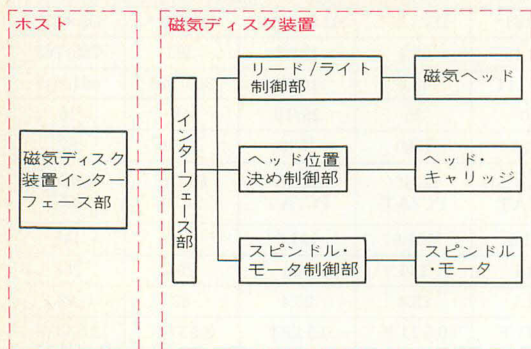
製　法
- 塗布型(磁性粉をバインダに溶かし塗装)
- 薄膜型(磁性体を直接メッキまたはスパッタ)

電子回路

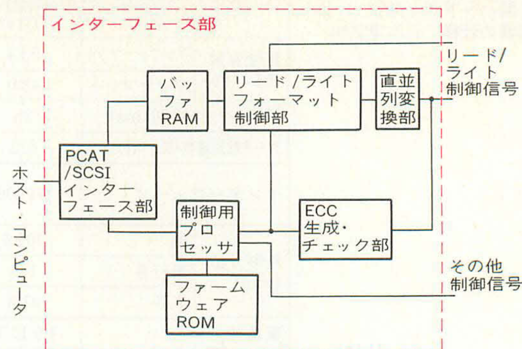
小型ハードディスク・ドライブの電子回路は，1 枚のプリント板の上に渾然一体となって配置されています．図 3 に示すようにその機能によりインターフェース部，スピンドル・モータ制御部，ヘッド・ポジショニング制御部，リード/ライト部に大別されます．

● インターフェース回路

インターフェース部は磁気ディスク・コントローラ



〈図3〉磁気ディスク装置の構成



〈図4〉エンベデッド・ディスクのインターフェース部の構成

もしくはホスト(パソコン本体など)と接続するための回路です.

5.25インチ以下のディスクで磁気ディスク・コントローラと接続するタイプとしてはST506/ST412インターフェースおよびESDIが一般的です.

これに対して最近では磁気ディスク・コントローラのLSI化が進み磁気ディスク装置の中にコントローラを取り込むタイプが主流になってきました.このようなタイプのディスクはエンベデッド・ディスクと呼ばれ,PC/ATインターフェースとSCSIが一般的です.

エンベデッド・ディスクのインターフェース部の構成を図4に示します.ホスト(パソコン本体など)からデータの記録や読み出し方法を指示するコマンドの解読,ホストとのデータの送受信および媒体上とのデータの送受信を行うバッファRAM制御,コマンドの実行結果の処理・報告を行う部分です.

ここでのリード/ライト制御信号などはST506やESDIの信号と同等の信号で,逆にいえばST506やESDIのインターフェースの後にコントローラがつながった形になっています.

▶ ST506/ST412インターフェース

1979年にシーゲート社からマイクロコンピュータ用のディスクとして5.25インチ・サイズのディスクST506が発売されました.

ST506はフロッピ・ディスク・インターフェースのデータ・レート(500Kbps)と回転速度を10倍(5Mbps)に,トラック数とヘッド数をそれぞれ倍にしたインターフェースです.このような関係からフロッピ・ライク・インターフェースとも呼ばれます.

ST506のシークはフロッピ・インターフェースの場合と同様,コントローラからのステップ・パルスによって直接動作します.

したがってシーク・スピードはコントローラからのステップ・パルスに依存するため,ドライブの性能が向上しても,それを反映することができません.

そこでシーゲート社はつぎのST412で倍トラック化(306シリンダ)と同時に磁気ディスク・ドライブの上にCPUをのせ,バッファード・シークと呼ぶモードを追加しました.これはシーク・パルスをまとめて送っておき,ドライブ側で最適になるように実際のシーク動作を独自に行うものです.

このバッファード・シークまでを含めたインターフェースをST506/ST412インターフェースもしくは単にST506インターフェースと呼びます.

▶ ESDI(エンハンスド・スモール・デバイス・インターフェース)

ST412はシリンダ数306とST506の倍のシリンダを持ちます.ディスクの容量を増やすためにST506インターフェースではシリンダ数とヘッド数を増やしました.

しかし,データ・レートはディスク・コントローラに依存するため5MbpsのMFMのまま変更されませんでした.また,MFM信号でインターフェースされ,データ・セパレータ(MFM→NRZ変換)はコントローラ側にありました.したがって,データ・レートが上昇するとインターフェース部の信号劣化が問題になります.

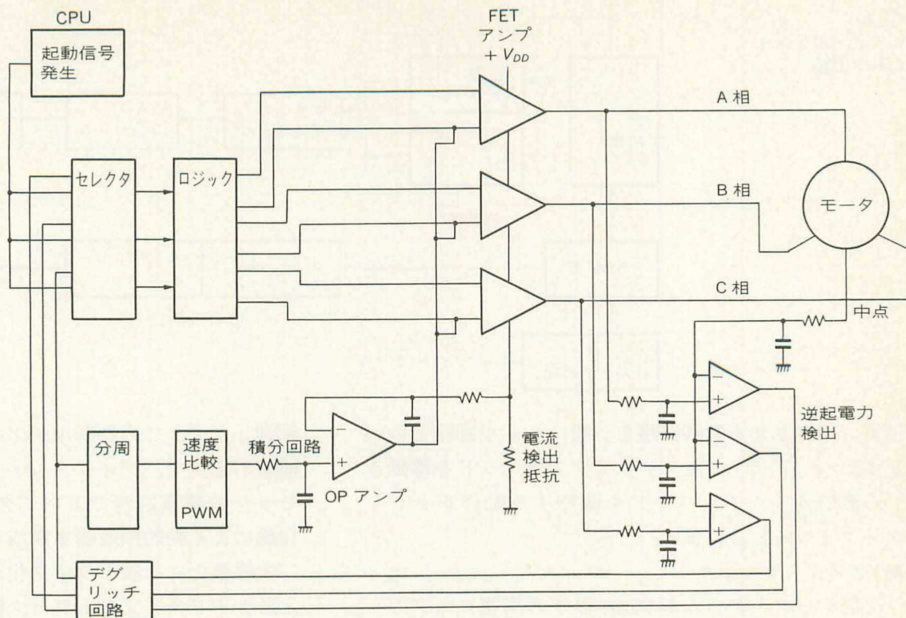
そこで,ESDIではデータ・セパレータをドライブ側に置き,NRZデータとクロックでインターフェースすることで10Mbpsから15Mbpsのデータ・レートまで扱えるようにしました.

また,シーク・パルスを送出する時間的なロスをなくし,その他のコマンドがやりとりできるようにシリアル・ラインでコマンド転送を行うようにしました.

▶ PC/ATインターフェース

ハードディスクに使用されているPC/ATインターフェースはIBMのパソコンPC/ATの増設スロットに使用されているインターフェースをハードディスク・ドライブ用に小型化したもので,ATA(ATバス・アタッチメント)として仕様化・標準化が進められています.

このインターフェースには現在のパソコンCPUの主流である80286/386に少ない部品を追加するだけで



〈図5〉
スピンドル・モータ回路のブロック図

ホスト回路が構成できるという特徴があり，このインターフェースがパソコンに標準装備されることでパソコンのハードディスク内蔵比率は一層高まるものと思われます．

▶ SCSIインターフェース

SCSI（スモール・コンピュータ・システム・インターフェース）は従来から使用されてきたSASIの機能拡張と仕様統一を目的として米国のANSIで制定され，現在では一層の機能拡張を行ったSCSI2の製品化が開始されています．

このインターフェースは十分な検討のもとに仕様化されており，磁気ディスク装置以外にも光ディスク装置，スキャナ，プリンタなどの機器を接続できる特徴があり，システム機能拡張には最適のインターフェースです．

インターフェース部には上記に示した機能のほかにエラー発生時のリトライ，ECCによる修正およびバッファRAMのデータ管理などの役割があります．エラー検出・訂正にはECCコードが使用されますが，そのコードも4バイトECCから現在は7バイトECCになり，さらに性能を向上させたリード・ソロモン・コードの使用も始まっています．このほか読み出しと同時に修正が行える方式も使われ始めています．

● スピンドル・モータ制御回路

ディスク・ドライブのスピンドル・モータとしては前述したDCブラシレス・モータが使用されています．この回路（図5）の機能はモータを短時間で起動させることと高精度で回転させることにあります．

モータの回転速度は現在ではディジタル回路で構成される速度検出回路から得られる速度信号と基準信号との差として得られる誤差信号により，モータ駆動電流を制御するフィードバック制御で行われます．

スピンドル機構のところで説明したように，小型化のために位置検出素子を省いたセンサレス・タイプのモータが使用されだしたため，モータ駆動には回転位置検出機能が必要となります．

通常はモータの回転によってモータ・コイルに発生する逆起電圧を検出することにより回転位置を検出しています．

モータが回転している場合には逆起電圧が発生するため上記の方法で回転位置は検出することができますが，起動時には逆起電圧が発生していないため簡単には位置検出ができず，モータが逆回転するなどの欠点があります．

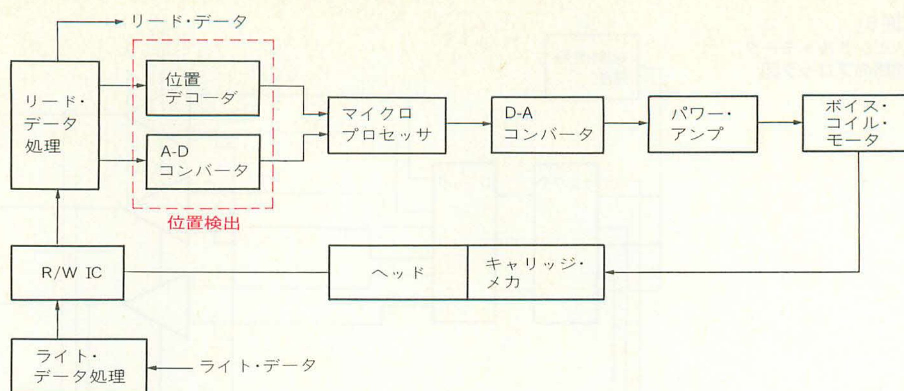
実際のディスク・ドライブではこの欠点を克服するためにモータが回転しない程度の短時間のモータ駆動を行った際のモータ・コイルに発生する電流または電圧をマイクロプロセッサで処理するなどして，起動時の回転位置を検出するといった工夫がなされています．

磁気ディスク装置の高速・大容量化の一手法としてディスク・アレイの開発が進められています．その効率向上のためには複数のディスク・ドライブを同期して回転させる必要があり，このスピンドル同期機能をサポートしたハードディスク・ドライブも出現しています．

● ヘッド・ポジショニング制御回路とサーボ

図6にヘッド・ポジショニング回路を示します．これは前述したヘッド・ポジショナ部の制御を行うもので，ヘッドを目標トラックに移動させ，目標トラック中央に保持する回路です．このような位置や速度を目

〈図6〉
ヘッド・ポジショニング回路



標値に追従させる制御回路を一般にサーボ回路といいます．ハードディスク・ドライブではヘッドを移動させる動作をシーク，ヘッドを保持する動作をトラック・フォローといいます．

▶ セクタ・サーボ

平均シーク・タイムが25ms以下の高速タイプのハードディスク装置はヘッドを動かす動力源であるアクチュエータにボイス・コイル・モータを使い，サーボ方式として専用サーボ面からサーボ情報によるデディケイテッド・サーボ方式と，データ面にサーボ情報を埋め込んだエンベデッド・サーボ方式が使われています．

ここでは，最近の小型ドライブに多用されているエンベデッド・サーボ方式のひとつであるセクタ・サーボについて説明します．

セクタ・サーボ方式は通常512バイトのデータとID情報などで構成されるセクタの間に，位置情報を記録したサーボ・セクタを埋め込む方式です．この方式では位置情報が媒体1回転につきセクタ数(30～70回)しか得られないため，アナログ制御よりマイクロプロセッサによるディジタル制御が適しています．

シーク制御は目標までの距離の関数としてあらかじめ設定した速度パターンと実際の速度を比較し，その差に比例した電流をボイス・コイル・モータに流すことによって速度を目標値に追従させます．

▶ 速度を位置情報やモータの電流値から算出

サーボ・セクタから得られる情報は位置だけなので，速度は位置情報やボイス・コイル・モータの電流値から算出します．ディジタル・サーボでは主として最新のセクタの位置情報とひとつ前のセクタの位置情報の差分で速度を計算します．

こうして求めた速度は全セクタでの速度と最新セクタでの速度の平均速度であり，およそセクタ周期の2分の1の時間遅れがあります．しかし，現代制御理論のオブザーバを使って速度を推定し，遅れを回避しています．

▶ トラック・フォロー

トラック・フォロー中は位置の誤差をなくすように制御します．この制御ループを安定化するために位相補償が必要で，ディジタル・サーボではマイクロプロセッサの演算処理でディジタルPID制御や現代制御理論による動的補償器を構成して安定化します．

大容量化には狭トラック化が必要であり，現在のディスク・ドライブのトラック幅は10μm前後となっています．このような狭トラックのディスク・ドライブでは極めて高精度のトラック・フォローが要求されます．

セクタ・サーボ方式の場合にはデータの読み書きに使用するヘッド自身で位置を検出するために，温度差によるヘッド間のトラック・オフセットを無視できるという特徴があります．説明を省略したデディケイテッド・サーボ方式の場合もトラック・フォロー時にはセクタ・サーボを一部併用するハイブリッド・サーボ方式が採用されつつあります．

今後はアナログ・サーボを上回る現代制御理論を活用したディジタル制御化と回路のLSI化がさらに進展すると思われます．

● リード/ライト制御回路

この部分はホストから転送されたデータをヘッドを介して媒体に書き込むライト部とヘッドを介して媒体の信号を読み出しホストに転送するデータ形式に変換するリード部から構成されます．各部の機能を図7にしたがって説明します．

▶ 変復調部

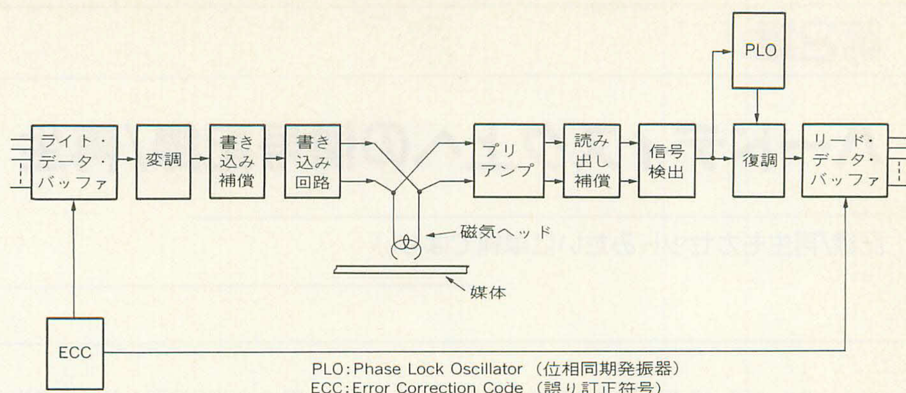
変調部はNRZ形式のデータを磁気記録する際，より多くの情報量が書き込めるデータ形式に変換する部分です．変調方式はFM→MFM→RLL(2,7)→RLL(1,7)と高効率化が図られてきています．

復調部は上記の変調方式で変調された読み出し信号をNRZ形式に戻す部分です．

▶ リード/ライト補償部

書き込み補償部は高密度記録の際に発生する磁化反転干渉による読み出し信号のピーク・シフトに対して書き込み時の磁化反転タイミングの微少補正を行う部分です．

〈図7〉
リード/ライト回路のブロック図



読み出し補償部は読み出し時の波形等価によるピーク・シフトの減小化を行う回路です.

▶ データ・セパレータ

データ・セパレータ部はパルス信号に整形した読み出し信号を基にリファレンス・クロックと呼ばれるリード処理時の基本クロックを作成するとともにリード・データのリファレンス・クロックへと同期化を行います.

リード/ライト回路の構成は小型ハードディスク・ドライブでは回路のLSI化が進行し,現在では2〜3チップ構成になってきています.近い将来には1チップLSIも現れると思われます.

以上,小型ハードディスク装置の概要を説明しました.技術面からその特徴を見ると,機械工学・制御工学・電子工学・システム工学などが複雑に交錯しながら集大成されています.

またLSIやマイクロプロセッサなどの利用によって,変調方式やエラー処理技術など高度な情報理論を応用したインテリジェント化が進んでおり,高信頼性維持に大きく貢献しています.

磁気ディスク装置は半導体メモリ,光ディスク装置の開発により,その製品寿命が幾度となく論じられてきましたが,予想を上回る記録密度の向上により大容量化・小型化・高速化を達成し,スーパーコンピュータ用からパーソナル・コンピュータ用まで多彩な製品を創出しており,今後とも発達し続けるものと考えられます.

第2章

ハードディスク上への信号記録/再生

記録/再生もカセットみたいに単純ではない

松村清明

ディジタル・データを磁気ディスクに記録し，それを再生してもとのディジタル・データを得る方法について述べます．

概要

記録再生系の概念を示したブロック図を図1に，また対応する各部の信号パターンを図2に示します．

HDC(ハード・ディスク・コントローラ)から出力されるNRZ(ディジタル・データをそのまま0，1の直列信号にしたもの)信号(a)を変調(b)し，このパルスを1/2分周して，パルスごとに書き込み電流方向を反転し(c)，この電流を磁気ヘッドに与えて磁化パターン(d)をディスク上に作ります．ディジタル磁気記録では通常このような磁化反転によって，データを記録します．

次に磁気ディスク上に書かれた磁化パターン(d)を磁気ヘッドで読み出すと(e)の再生波形が得られます．実際には図の波形より時間軸方向右にずれた波形が磁気ヘッドから出力されますが，ここでは説明のため磁化反転位置と再生波形のピークを揃えました．(f)，(g)の波形も同様にディレイがない場合の波形を示しています．

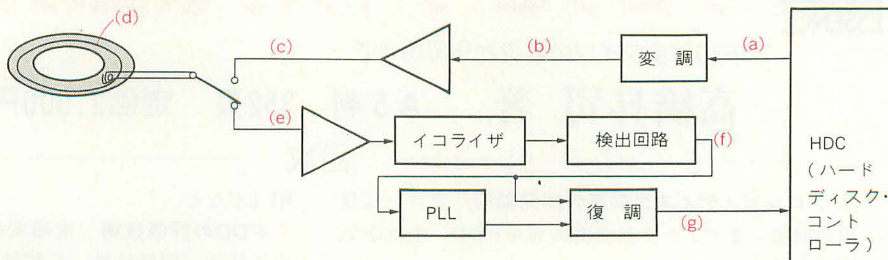
この再生波形を整形し，ピークを検出することで検出パルス(f)が得られます．このパルスを基にPLLで記録時のクロックを再生し，復調回路によって復調データ(g)を得て，HDCへ入力します．

これらの回路を総称して記録再生系もしくは信号処理系と呼びます．

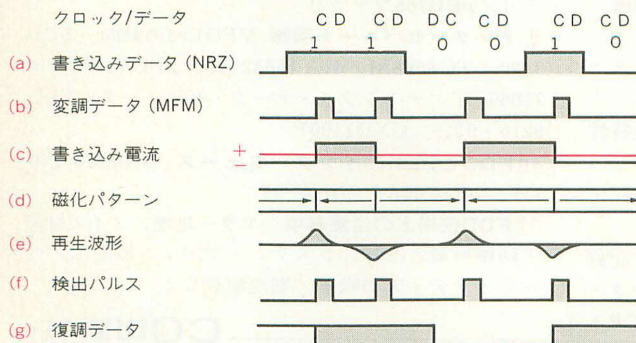
以降の項で各ブロックについて説明します．

変復調方式

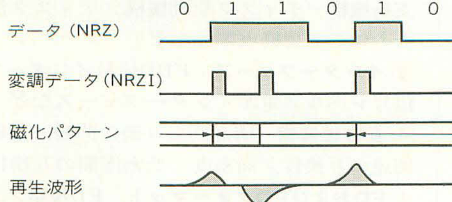
ディジタル・データを記録媒体に記録できるような形に変換することを変調といい，その記録をもとのデ



〈図1〉記録再生系のブロック図



〈図2〉各ブロックにおける波形



〈図3〉NRZI方式

ータに戻すことを復調といいます．

● NRZI方式

最も単純な変調方式であるNRZI方式を図3に示します．

NRZI(Non Return Zero Inverted)方式は1のときに磁化方向を反転し，0のときはそのままにする方式です．

この方式は単純ですが，0が連続した場合には磁化反転が発生しません．したがってそんな場合には再生波形から記録時のクロックをつくりだすこと(セルフ・クロッキング)ができません．

そこで，一般的には別にクロックを記録しておき，その信号を別のヘッドで読み出してクロックを作ります．しかし，この方法ではクロックの位相とデータの位相が合っていることが必要になります．

ビット・レートが低いときにはNRZI方式を変調方式として使用したディスクもありましたが，ビット・レートの上昇とともに，データとクロックの同期が困難になり，セルフ・クロッキング可能な(再生波形からデータとともにクロックが再生できる)FM(Frequency Modulation)方式やMFM(Modified Frequency Modulation)方式が開発されました．

● FM方式とMFM方式

図4にこれらの方式を示します．データ/ビットはクロック・ビット(図中のC)とデータ・ビット(図中のD)の2ビットに変換されます．FM方式はすべてのクロック・ビットでパルスを発生させ，データ・ビットではデータが1の場合のみパルスを発生させる方法で，クロック・ビットが必ずあるため再生波形からクロックを容易に再生できます．しかし，1ビットに対して2回磁化反転が発生するので，最小磁化反転間隔がNRZI方式の1/2になってしまいます．

MFM方式は磁化反転間隔を落とさずにセルフ・クロッキングができる方式として開発されました．この方法はデータが1の場合にはデータ・ビット位置にパルスを発生し，0が2ビット以上連続した場合には2ビット以降の0のクロック・ビット位置にパルスを発生します．これにより，最小磁化反転間隔はデータ・ビットの間隔となりNRZI方式と等しくなります．ただし，クロック・ビットの発生が0が連続した場合に限られるため，クロック・ビットのみからクロックを再生することはできず，データ・ビットを含めデータ・レートの倍のレートで同期クロックを発生する必要があります．

そこで，このクロックをデータ・ビットに同期したタイミングで1/2分周する必要があります．そして図5に示すようにMFMの変換方式では発生しないようなビット列(ミッシング・クロック)をマークとしてビット同期を取ります．またこのマークは後述するデータ・フォーマットの中のアドレス・マークとして使用され，バイトの同期を取るためにも使用されます．

MFM方式は最小磁化反転間隔が大きい割に変換方式やクロック再生が容易であるため，フロッピ・ディスクや小型ハード・ディスクで多数使用されています．

● RLL方式

ハード・ディスクの高容量化にともない固定ディスクのようにメディア互換性を必要としない用途においては，より高密度な変換方式を使用するようになってきました．そこで開発されたのがRLL(Run Length Limited)方式です．RLL方式には多くの方式が提案されていますが，現在最も多く使用されているのがRLL(2, 7)で，多く採用されつつあるのがRLL(1, 7)です．

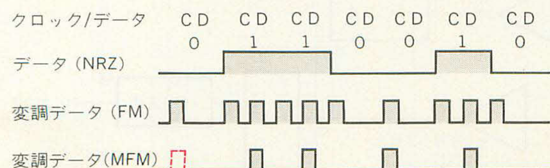
ここで，RLL(x, y)としたときのxとyはコード化されたときの1と1の間に出現する0の数の最小と最大です．この表現を用いればMFMはRLL(1, 3)ということになります．これらのデータとコードの変換表を表1に示します．またRLL(2, 7)の変換例を図6に示します．

MFMの場合には0が始まるときを除いてデータ1ビットがコード2ビットに対応します．しかし，RLL(2, 7)の場合にはデータの並びによって2ビットもしくは3ビットが，4ビットもしくは6ビットのコードに対応し，RLL(1, 7)においては先行するビットも含めると2ビットから5ビットのデータをもとに1.5倍にコード化されます．

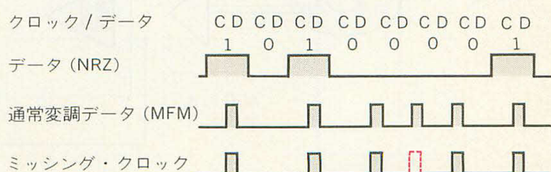
このことは再生データにひとつの誤りがあった場合，復調において複数のビットの誤りとして出現する(エラーが伝播する)ことを意味します．したがって，MFM，RLL(2, 7)，RLL(1, 7)の順にエラーの伝播ビット数が多くなります．

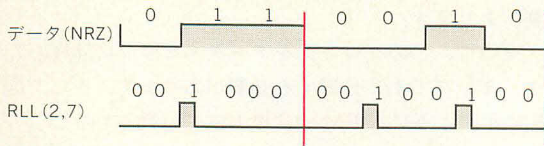
RLL(2, 7)の最小磁化反転間隔はMFMの1.5倍であるため，線記録密度(磁化反転密度)を一定とするとMFM方式で記録する場合と比較して1.5倍のデータ

〈図4〉FM，MFM方式



〈図5〉ミッシング・クロック





〈図6〉RLL 変換例

を記録できます．ただしそれだけ符号の冗長度が少ないので，ミッシング・クロックのようなビット同期方式は使えません．

そこでソフト・セクタ(第3章で解説)においてはセクタ・マークとして24ビットから32ビット程度のイレーズ・ギャップを作り，これを再生パルスのギャップから検出する方法が一般的です．またシンク領域においてビット同期し，アドレス・マークのデータでバイト同期します．

各種変調方式のパラメータを表2に示します．磁化反転間隔は，先に述べたものをまとめて表しています．検出窓幅はビット列の位置を検出するための窓の幅で，大きい程復調が容易であることを示します．NRZI 方式はセルフ・クロッキングできないので例外です．

MFM 方式とRLL(2, 7)はデータ1ビットあたりコード2ビットであるため，どちらも$0.5T$の窓幅になります．これに対し，RLL(1, 7)の磁化反転間隔は$1.33T$でRLL(2, 7)の$1.5T$より若干悪くなりますが，RLL(1, 7)はデータ2ビットがコード3ビットになるので2/3の窓幅を持ち，復調が容易であるといえます．これが最近高データ・レートのディスクでRLL(1, 7)の採用が多くなってきた理由です．しかし，先に述べたようにエラーの伝播が大きいため，RLL(2, 7)を採用し続けるメーカも多いようです．

〈表1〉RLL 方式変換表

方式	データ	コード
MFM〔RLL(1,3)〕	0 1	×0 01
RLL(2,7)	000 10 010 0010 11 011 0011	000100 0100 100100 00100100 1000 001000 00001000
RLL(1,7)	01 10 11 0001 0010 0011 0000	×00 010 ×01 ×00001 ×00000 010001 010000

×＝先行するビットの補数

〈表2〉各種変調方式のパラメータ

方式	磁化反転間隔		検出窓幅
	最小	最大	
NRZI	T	∞	T
MFM	T	$2T$	$0.5T$
RLL(2,7)	$1.5T$	$4T$	$0.5T$
RLL(1,7)	$1.33T$	$5.33T$	$0.67T$

信号検出

磁化パターンを磁気ヘッドで読み出し，パルス化するまでの信号検出部分のブロック図を図7に示します．対応する各点の信号波形を図8に示します．

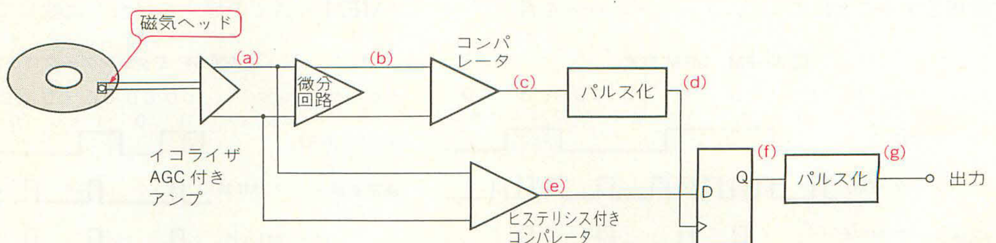
磁化パターンを磁気ヘッドにより読み取り，再生した波形(a)のピークが，磁化パターンの変化点を表します．そこでこれを微分(b)し，コンパレータ(c)で0クロス点をエッジとして取り出します．このエッジをパルス化してパルス列(d)を得ます．

ここで図8にあるように磁化反転間隔が大きい場合，再生波形のピークとピークの間に振幅のないタイミングができます．これを微分すると微分波形はゼロ近くでノイズ性の信号となり，コンパレータ出力にノイズが発生し，疑似パルスの発生をまねきます．

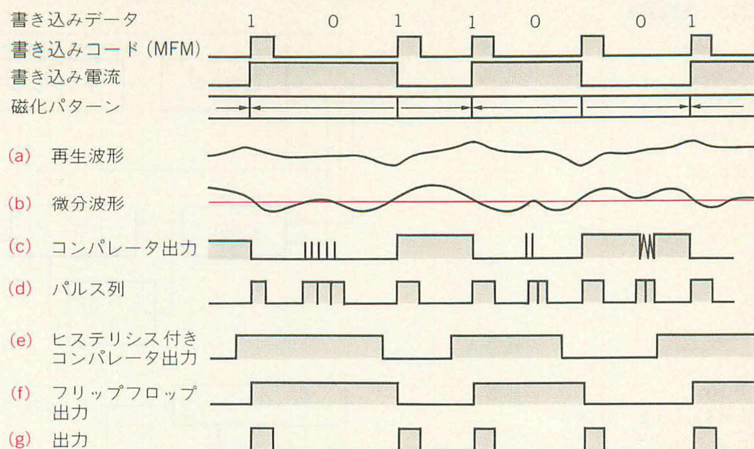
このノイズを除く方法は種々考案されていますが図7，図8では原波形スライス方式と呼ばれる回路を説明しています．

再生波形(a)をヒステリシスのあるコンパレータに通してコンパレート出力(e)を得ます．この信号は書き込み電流波形を少し早めたような波形となります．これを(d)パルス列でサンプリングすると，真のパルスの中間あたりにある疑似パルスではヒステリシス・コンパレータ出力が直前の真のパルスのときと同相であるため，(f)のような波形が得られます．これをパルス化して必要な波形(g)を得ることができます．

〈図7〉信号検出部ブロック図



〈図8〉信号検出部の波形



復調回路

信号検出部から出力されたパルス列をHDCに入力するためにNRZに変換する部分を復調回路といいます．この部分のブロック図を図9に示します．

復調回路は大きくPLL(Phase Locked Loop)とデコーダに分かれます．PLLの部分はリード・データと位相同期したVCOクロックを作ります．このクロックはリード・データのサンプリング・クロックになります．

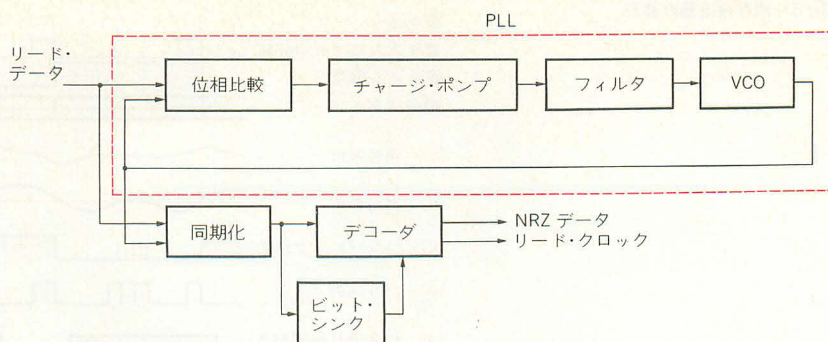
ディスク用のPLLのロックする周波数はディスクの回転誤差である±1％ぐらいの精度で一定周波数です．重要なことはID部やデータ部の前のシンク領域(3章)で，リード・データとVCOの位相差を急速に引き込むことと，データ領域でゆるやかな周波数変動に追従することです．変調方式によって種々の方法がありますが，PLLについてはここでは述べません．

ハード・ディスク用VFOチップの仕様書などを参照して下さい．

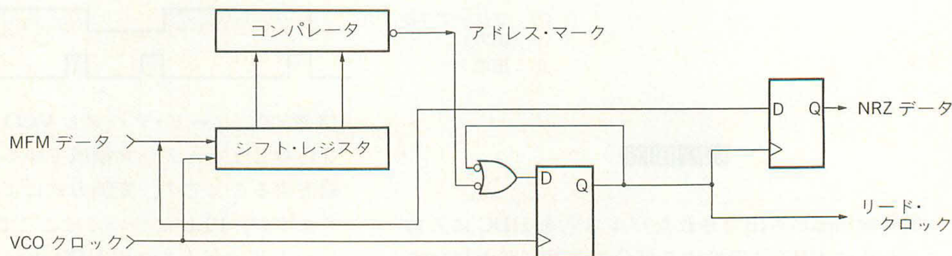
デコーダの部分は変調方式によってかなり異なります．図10にMFMの場合の，図11にRLL(2,7)の場合のビット・シンクおよびデコードのブロック図を示します．

図10のMFMの場合には，VCOクロックで同期化されたMFMデータをサンプリングし，ミッシング・クロックによりつくられたユニークなアドレス・マークを検出します．これは，VCOクロック(データ・

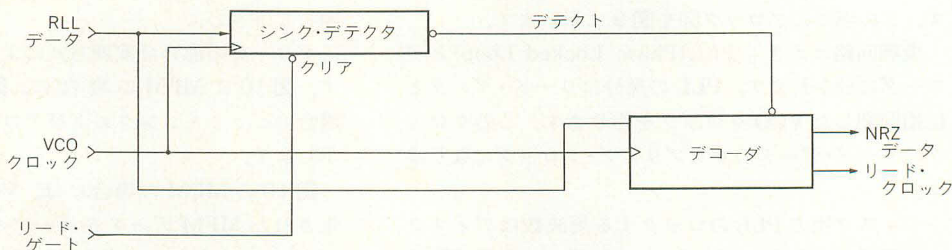
〈図 9〉 復調回路



〈図 10〉 MFM のビット・シンク



〈図 11〉 RLL(2,7) のビット・シンク



クロックの倍)で MFM データをサンプリングすることで，一意に検出できます．検出ビットにより VCO の 1/2 分周を調整し，リード・クロックを作ります．

また MFM の場合には 1 のデータに対して，必ずビットが立っているので，MFM データを単純にリード・クロックでサンプルすることにより，NRZ データをつくることができます．

図 11 の RLL(2,7) の場合はもう少し複雑です．まず VCO クロックに同期化された RLL データとともにシンク領域と同時に立ち上がるリード・ゲート信号を使います．リード・ゲートが立ち上がって PLL がロックすると RLL データはシンク・パターン(0001 もしくは 001 の連続)になります．一定回数(たとえば 8 回)連続してシンク・パターンが入力されるとシンク・デテクタはシンクを検出したということでデテクト信号を出力します．

この信号に同期してデコーダはリード・クロックをビット同期し，直前 4 ビットの RLL データをデコードして NRZ 信号をつくります．

以上述べた各ステップにより磁化データをリードして NRZ 信号を復調し，書き込み時に HDC から出された NRZ 信号と同等の信号列を得ます．

◈参考・引用*文献◈

(1)*Max Roth：コストを抑えて磁気ディスクの容量を増やすためには効率の良い信号符号の選択が重要，日経エレクトロニクス，1985.7.15，pp.289-297.
(2) 伊藤陽之助他：ディジタル磁気記録技術，トリケップス，1986.
(3) National Semiconductor，DP8462 2，7Code Data Synchronizer.
(4) WESTERN DIGITAL，WD10C23，1989.
(5) National Semiconductor，DP8469 Synchronizer，1989.
(6) Adaptec，inc，AIC-270，1986.
(7) 日立，HD153009 磁気ディスク用 2-7 符号変復調 IC．1987.
(8) 日立，HD153014F データセパレータ，1990.

代表的なHDC

松村清明

● μPD72061(日本電気)

μPD72061はμPD7261をCMOS化したハードディスク・コントローラ(HDC)です．ハード・セクタ・ディスク(SMDインターフェース)，およびソフト・セクタ・ディスク(ST506インターフェース)の両方を取り扱うことが可能です．

▶特　徴

- μPD7261A/Bとコンパチブル
- CMOS
- 2種のディスク・ドライブにインターフェース可能
  ハード・セクタ・ディスク(SMDインターフェース)
  ソフト・セクタ・ディスク(ST506インターフェース)
- プログラマブル・トラック・フォーマット
  データ長：128～4095バイト/セクタまで指定可能
  各種ギャップ長可変
- データ転送速度
  24Mbps：μPD72061-24(ハード・セクタ)
  12Mbps：μPD72061-12(ハード・セクタ)
  　　　　μPD72061-24(ソフト・セクタ)
  6Mbps：μPD72061-12(ソフト・セクタ)
- ディスク・ドライブ：8台まで接続可能(SMDインターフェース)
  　4台まで接続可能(ST506インターフェース)
- パラレル・シーク動作可能
- マルチ・セクタ機能
- マルチ・トラック機能
- データ・スキャン，データ・ベリファイ機能
- CRC発生，チェック機能
- ECC発生，チェック，訂正機能
  11ビットまでのバースト・エラーを訂正可能
- DMAデータ転送

● μPD7262(日本電気)

μPD7262ESDIC(Enhansed Small Device Interface Controller)はシリアル・モードESDIのハードディスク・ドライブを7台まで制御可能なハードディスク・コントローラです(図A)．

▶特　徴

- シリアル・モードESDI対応
- 7台までのドライブを制御可能
- ハード・セクタ，ソフト・セクタ混在制御可能
- プログラマブル・トラック・フォーマット
  データ長可変：128～65536バイト/セクタ
  ギャップ長可変：インタセクタ・ギャップ
  　PLLシンク
  　スピード・トレランス・ギャップ
- セクタ開始信号選択
  セクタ・パルス/AMF信号
- データ転送レート
  max．18Mbps：μPD7262-18
  max．12Mbps：μPD7262-12
- 23種類のディスク・コマンド
- パラレル・シーク動作
- マルチ・セクタ/マルチ・トラック/マルチ・シリンダ機能
- インプライド・シーク機能
- データ・スキャン/ベリファイ機能
- エラー・セクタのデータ読み出し機能
- ECC/CRC選択可
  ECC：32/56ビット
  CRC：16ビット
- Nチャネル MOS

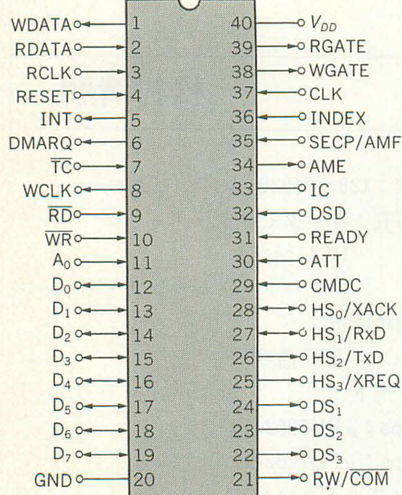
● HD63463(日立)

HD63463は16ビット・プロセッサHD68000の周辺LSIとして開発されたCMOSのHDCです．SRAMを内蔵し，低速インターフェースしか持たないホストにも接続できます(図B)．

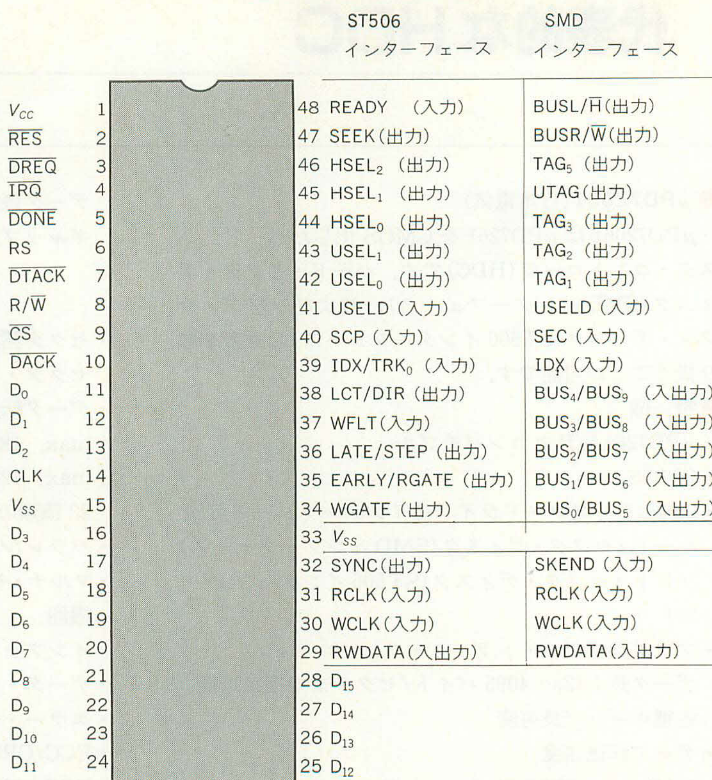
▶特　徴

- 2種類のディスク・ドライブ・インターフェース可能
- ハード・セクタ(ST506インターフェース)
- ソフト・セクタ(SMDインターフェース)
- データ・バス幅選択が可能(16/8ビット)
- シリアル・データ転送レート(ハード・セクタ20Mbps，ソフト・セクタ10Mbps)
- データ・バッファ内蔵
- 256バイト×2面のSRAMを内蔵し，ホスト側とドライブ側の互いに独立なデータ転送の実現
  DMAなしでインターフェース可能
- ディスク・データをデータ・バッファ内で自動訂正し，訂正済みのデータをホスト・システムに転送(11ビットまでの連続したエラー訂正が可能)
- 22種類のディスク・コマンド
- CRC発生，チェック，32ビットECC発生，チェック，訂正
- CMOSプロセス

〈図A〉 μPD7262のピン配置



〈図B〉 HD63463のピン配置



● MB89342A(富士通)

MB89342A/89342AH はローカル・メモリのコントロール機能を持つ HDC です．ローカル・メモリがある場合，DMA を必要としません．ESDI のハードディスク・ドライブを制御します．

▶特　徴

- 高速データ・レート
  - MB89342A　　10Mbps(max. 12Mbps)
  - MB89342AH　　15Mbps(max. 18Mbps)
- ローカル・メモリ制御可能(アービトレーション付き)
- ESDI　REV. B　仕様サポート
- ID バイト長　5 バイト/6 バイト可変
- 8 バイト FIFO による転送タイミング制御の軽減
- アドレス・レジスタ構造による簡素なインターフェース
- オート・リトライ・モード，ノン・リトライ・モード可
- オート・インターリーブ・フォーマット可(ハード・セクタ)
- CMOS

第3章

記録/再生信号の変調と復調

マイクロシーケンサと訂正符号は信号記録技術の基本

松村清明

前章で述べた記録再生系に対して，NRZ信号でインターフェースすることによりセクタ単位のデータとしてリード/ライトする部分について説明します．

ディスクのフォーマット

ディスク上に書かれるデータはインデックス信号(1周ごとに1回出る信号で，ディスクの回転位置を示す信号)から始まり，次のインデックス信号で終わる一周の記録領域を，セクタと呼ばれる複数個の小領域に分割して使用するのが一般的です．この小領域への分割方法として，セクタ・パルスといわれるパルスを発生させて一周をハードウェア的に分割するハード・セクタ方式と，記録領域のデータとしてユニークなデータを書き込むことにより，そのデータを検出してセクタの区切りとするソフト・セクタ方式の二方式があります．

この二方式の差はIDブロック部の構成にあります．ハード・セクタ方式のセクタ・フォーマットを図1に，ID部の差異を図3～図7に示します．

ハード・セクタ方式におけるセクタはセクタ・パルスで始まります．ヘッド・スキャッタから始まり，CRC，PADまでがID部で，このセクタのアドレスを示しています．ID部はトラック・フォーマットなどのフォーマット時を除き，通常のライト処理時に書かれることはありません．ID部についてはほかの方式とともに後述します．

これに対しデータ・スキャッタから始まりECC，PADまでがデータ部で，PLLシンクからPADまでがライト処理において書かれます．したがって書き継ぎ点(部分的に上書きされるためにリード・パルスに連続性がなくなる点)がデータ・スキャッタ部とISGAP(セクタ間ギャップ)に発生し，PLLのロックが乱れます．

●データ部

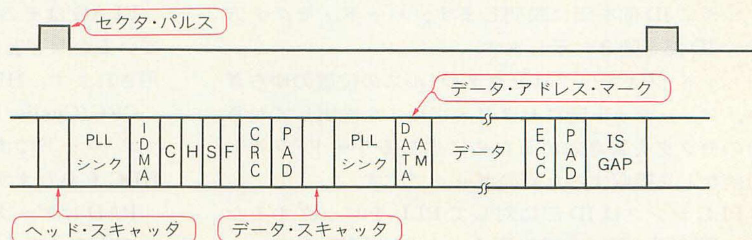
データ部を順に説明します．

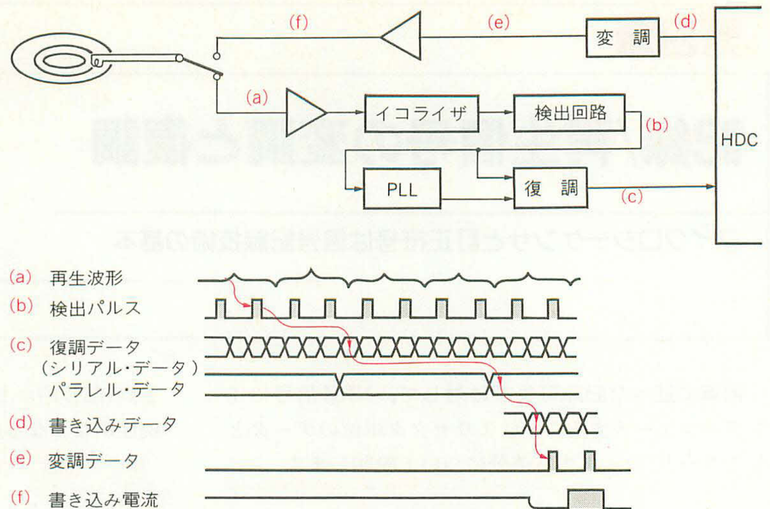
データ・スキャッタは書き込み時にそれまでID部をリードしていたヘッドに対し，書き込みを開始するときの時間差を吸収するためのギャップです．この時間にはリード・ライト・アンプなどのアナログ的ディレイのほかに変調/復調に必要なビットのディレイも含まれます．

この関係を図2に示します．ヘッドから読み出された再生波形(a)は検出回路のディレイで若干遅れて検出パルス(b)になります．復調器の中ではデコードのために4ビット位のデータを使用するので4ビット遅れで復調データ(c)になります．このデータ(シリアル・データ)はHDC(ハードディスク・コントローラ)の中でシリアル-パラレル変換されるため，最大で8ビット遅れます．つぎに書き込みをスタートして書き込みデータ(d)を出力しますが，この間にはディスク・コントローラの内部処理時間が必要なので少なくとも数ビット分の遅れになります．書き込みデータは変調器で変調され，変調データ(e)になります．ここで1～2ビット遅れ，ヘッドを通して書き込まれます．

このように変調方式がRLLの場合には10ビットから20ビット程度の遅れが発生します．したがってID部をリードしていた状態から直ちにデータ部を書き始めても，ID部とデータ部の間に隙間ができてしまいます．これを吸収するために少し大きめに開ける

〈図1〉
ハード・セクタ方式のフォーマット





〈図2〉リード・ライト・ディレイ

ギャップがデータ・スキャッタです．

PLLシンクは書き継ぎ点で乱れたPLLのロックを回復し，RLLの場合にはビット同期を確立するために用いられます．通常10バイトから20バイト程度の長さになります．

データ・アドレス・マークは，これ以降のデータを読むにあたってのバイト同期を取るためのもので，MFMの場合にはミッシング・クロックのアドレス・マークを，RLLの場合には1で始まる特定のパターンを使用します．

データはセクタ・サイズ分のデータから構成されます．ECCはデータ領域の誤り検出，訂正を行うためのコードで，32ビットもしくはそれ以上の長さを持ちます．

PADはライト時，ECCを終了した時点で変調回路の中に残っているコードやリード時にECCを最後まで安定に読むためのヌル・データです．ライト時はPADまで書き，リード時はECCまで読みます．

ISGAPはつぎのセクタの始まりまでのギャップで，データ部を書くときの位置のゆらぎや回転速度誤差による長さの差を吸収するためのギャップです．HDCによってはつぎのセクタ処理に入るまでに時間がかかるため，それを確保するためにISGAPを多めに取る必要があるものもあります．

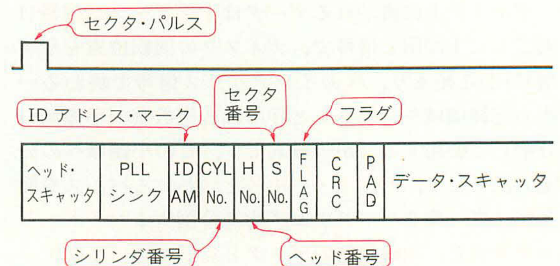
● ID部

つぎにID部を順に説明します．ハード・セクタ方式のID部を図3に示します．

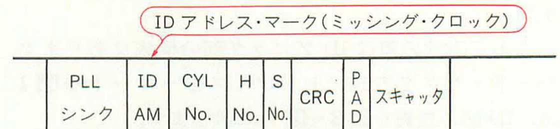
ヘッド・スキャッタはセクタ・パルスの位置のゆらぎや，フォーマット時にセクタ・パルスを検出してからそのセクタを書き始めるために生じるリード/ライトの差などを吸収するためのギャップです．

PLLシンクはID部に対してPLLをロックするための領域で，データ部のPLLシンクと同等です．

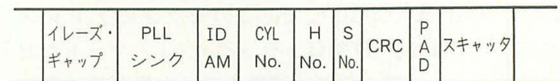
〈図3〉IDブロックの詳細



〈図4〉ソフト・セクタ方式のIDブロック(MFM)



〈図5〉ソフト・セクタ方式のIDブロック(RLL)



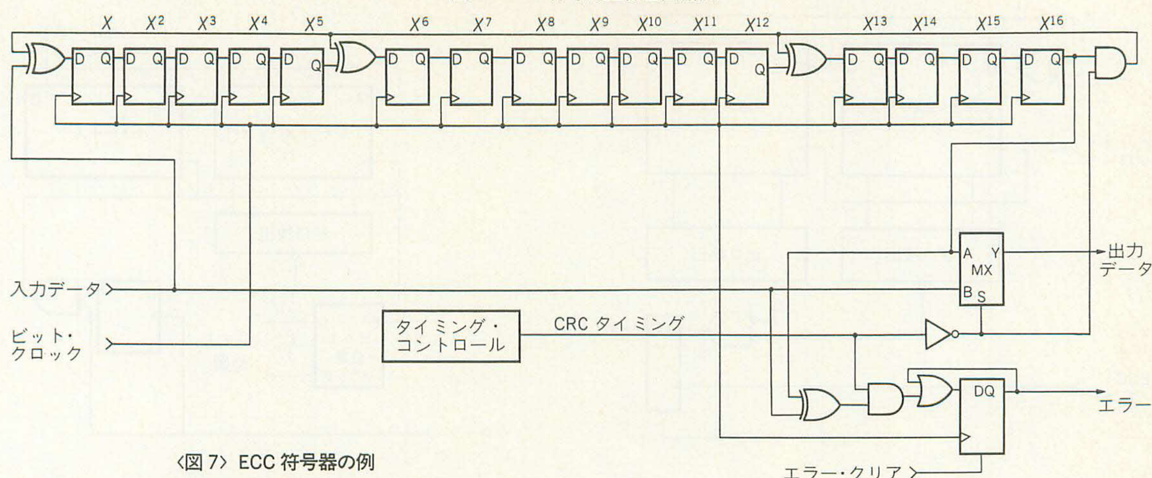
IDアドレス・マークはハード・セクタ方式では1で始まる特定パターンが使用されます．Cyl. No, HNo, SNoにはそれぞれこのセクタのシリンダ，ヘッド，セクタの各番号がセットされています．

FLAGはそのセクタが正常セクタか代替処理されているかなどといった付帯的情報を設定するために使用されます．HDCによってはないものもあります．

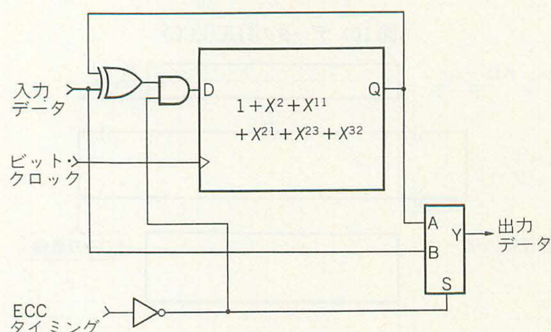
CRC(Cyclic Redundancy Check)はID部のチェック・コードです．データ部と同様にECCを使用するHDCもあります．

PADはデータ部のPADと同じ意味ですが，ID部のみを書くことはないのでフォーマット時にはデー

〈図6〉CRC符号発生検査回路例



〈図7〉ECC符号器の例



タ・スキャッタと一体で処理されます.

MFMのソフト・セクタ方式のIDブロックの例を図4に示します.

ハード・セクタ方式と異なるのはIDアドレス・マークの部分です.ソフト・セクタ方式ではセクタの途中から読み始めてもセクタの先頭を見つける方法が必要です.そこでIDアドレス・マークとしてほかのパターンと明確に区別できるユニークなパターンをミッシング・クロックによって作っています.このような方法を使うことでアドレス・マークはどこから読み始めても検出できるわけです.

つぎにRLLの場合のソフト・セクタ方式について図5で説明します.

RLLはMFMほど冗長ではないので,ミッシング・クロックのようなユニークなパターンを作ることは困難です.そこでユニークなビット・パターンとして24ビットから32ビット程度のイレーズ・ギャップを作り,これを検出することでアドレス・マークとします.

これらソフト・セクタの方式はアドレス・マーク部以外はハード・セクタの場合と同様で,各方式によるデータ領域の差はほとんどありません.

誤り検出訂正

磁気ディスクの場合,メディア上の欠陥や記録再生系におけるジッタ,ノイズなどの原因によって必ずしも書かれた通りのデータが読まれるとはかぎりません.

したがって通常は誤り検出のためのコードをID部およびデータ部に付加することにより,読み出しにおいて誤りがあったかどうかを検出します.フロッピ・ディスクの場合にはID部,データ部ともにCRCが使用されます.これに対し,ハード・ディスクの場合にはデータ部に対してファイア符号と呼ばれるECCを使用するのが一般的です.

これらの誤り検出方法はどちらも巡回符号と呼ばれる符号を用いるもので,入力データ $V(x)$ を生成多項式 $G(x)$ で割ったときの剰余を検査ビットにします.CRCとして一般に用いられる生成多項式 $G(x)$ は,

$$X^{16}+X^{12}+X^{5}+1$$

です.CRCコード発生,検査のための実際の回路を図6に示します.

入力データ(X^0の項)は割り算の結果 X^{16} と排他的論理和を取り,それを入力します.X^5 と X^{12} の項も X^0 と同様です.

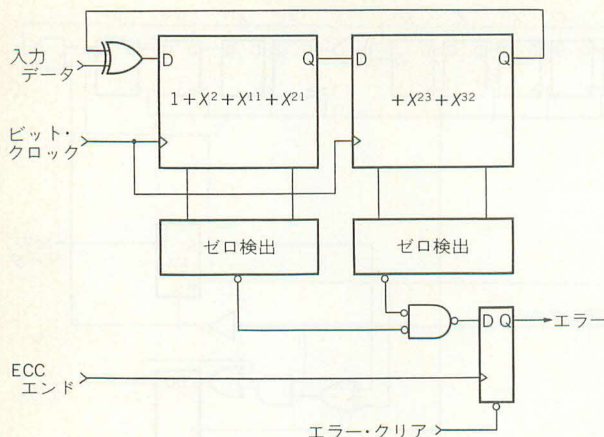
コードを発生する場合,データ領域では入力データがそのまま出力データとなります.CRC領域では X^{16} のフィード・バックをマスクしてその時点で残っていた数(剰余)を出力データとします.

コードを検査する場合,データ領域では剰余を作るだけです.CRC領域において入力データと剰余を比較し,異なっていればエラーとします.

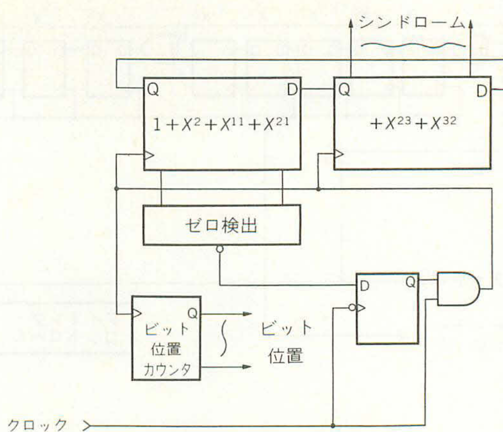
つぎにECC符号器の例を図7に示します.

32ビットのECCの場合,生成多項式は,

$$(X^{21}+1)(X^{11}+X^{2}+1)$$



〈図8〉誤り検出(ECC)



〈図9〉シンドローム生成

です．剰余回路はこれを展開した形で実現するのが普通です．上の式を展開すると，

$$X^{32}+X^{23}+X^{21}+X^{11}+X^{2}+1$$

となります．ECCの生成の場合は多項式生成部分が異なる以外ほとんど同じで，データ領域の終わりに剰余をECCデータとして出力します．

つぎに図8で誤り検出時の動作を説明します．データ領域とそれに続くECC部の終わりまでをECC生成回路に入力し，その時点で剰余が0(X^1〜X^{31}がすべて0)ならば誤りはありません．

もし0でなかった場合には誤りを含むことを表します．

この場合は図9に示すように入力データを0として逆方向に巡回させ，下21ビットがすべて0になるところまでシフトします．全入力ビット分シフトしても下21ビットがすべて0になることがない場合には訂正不能エラーです．下21ビットがすべて0になったときの上11ビットがシンドローム，シフトした数がECC領域もふくめた最終ビットからシンドローム・データの最下位ビットまでのビット数です．ここでシンドロームとは病気の症候という意味から来た言葉で，誤りの症状を表すデータであることからこのように呼ばれます．

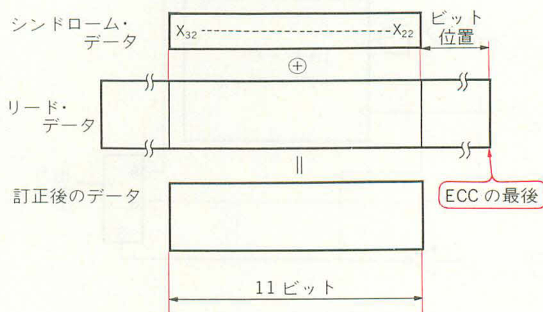
図10に得られたシンドロームおよびビット位置からデータ訂正する方法を示します．

このようにデータとシンドロームの排他的論理和を取ることによってデータを訂正します．

ここでは単純にシフトしていく方法でシンドロームを得る方法を述べましたが，この方法では最悪時，全ビット数分シフトする必要があります．

現在のハードディスクでは磁気媒体に欠陥がある領域をあらかじめサーティファイして検出しておき，代替処理をして使用します．したがって通常の使用でECCによる訂正が発生することはまずありません．

〈図10〉データの訂正(ECC)



これに対して光磁気ディスクなどではエラーが発生することを前提として処理され，リアルタイム訂正が必要になるのでリード・ソロモン・コードが使用されています．

今後は磁気記録方式においても記録密度の上昇とともにこのようなリアルタイム性のある誤り訂正能力を持ち，誤りが含まれていることを前提としたシステムが出現してくるかもしれません．

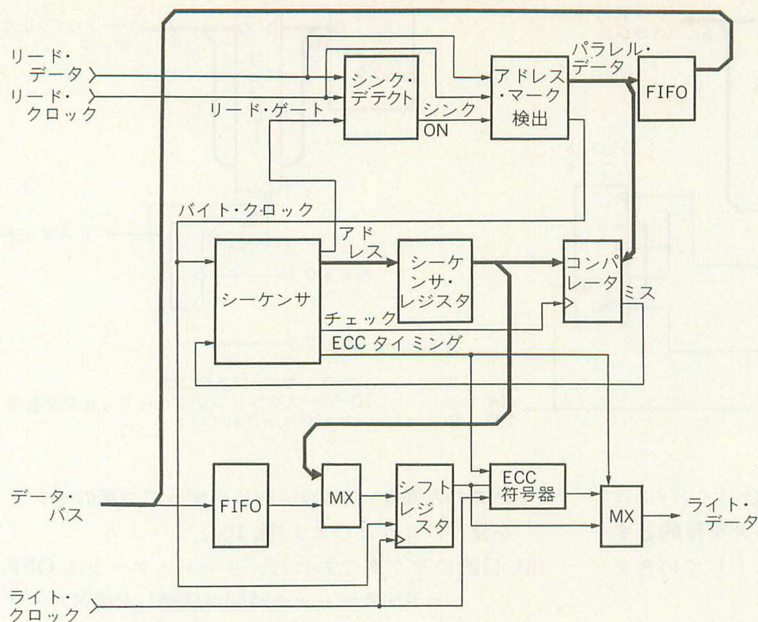
フォーマッタの動作

フォーマッタは通常マイクロ・プログラム・シーケンサで組まれており，シーケンス自体はマイクロ・プログラムにより実行されています．フォーマッタの概略ブロック図を図11に示します．また各部の動作を以下で順に述べていきます．

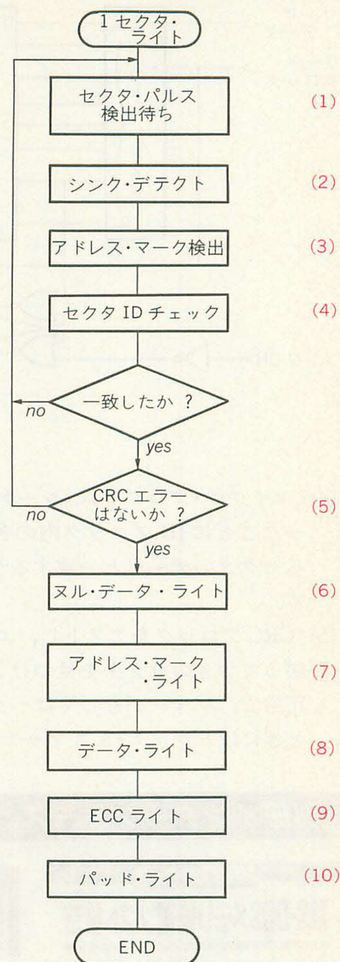
前述のようにトラックはセクタに分割されており，セクタごとに同一の処理を行うので1セクタの処理について説明を加えていきます．

また，セクタはID部とデータ部により構成されます．ライトとは目的のID部を見つけ，その後に続くデータ部にデータ・ブロックを書き込むことです．目的のID部とは，ID部の中のシリンダ，ヘッド，セク

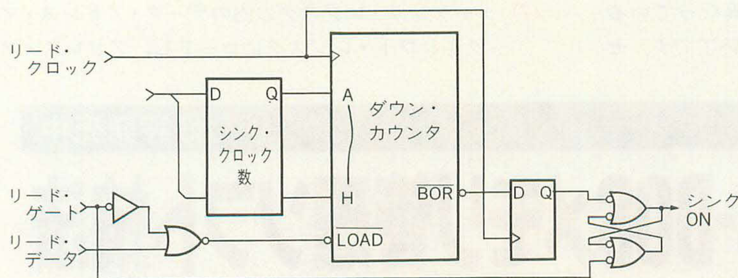
〈図 11〉フォーマッタの概要



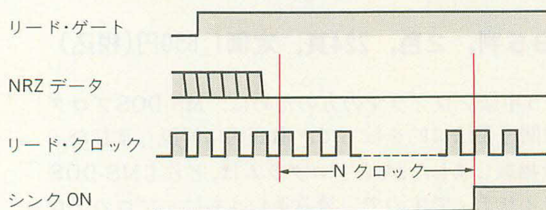
〈図 12〉ライト動作のシーケンス



〈図 13〉シンク・デテクト



〈図 14〉シンク・デテクト・タイミング



タがすべて目的とするセクタの値と合う ID 部を指します．ライト動作のシーケンスの流れを図 12 に示します．

(1) ID 部をリードする準備ができると，セクタ・パルスを見つけにいきます．

(2) セクタ・パルスが見つかるとヘッド・スキャッタ分待ってリード・ゲートを ON し，シンク・パターンを検出しにいきます．

ヘッド・スキャッタとは図 2 で述べたようなリード・ライト間の総合的なディレイにより発生するライト時とリード時の時間差を指すものです．

図 13 にシンク・デテクトの回路を示します．

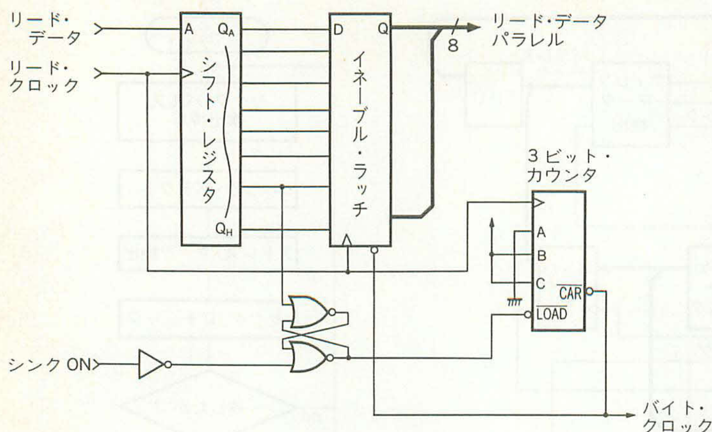
リード・データが 1 かリード・ゲートが OFF のとき，あらかじめレジスタに設定されているシンク・クロック数がダウン・カウンタにロードされます．したがってリード・ゲートが ON してから，0 のリード・データがシンク・クロック数分続くとシンク・デテクトとされ，シンク・オンがアクティブになります．これによりVFO がリード・データに同期したことを確認します．このときのタイミングを図 14 に示します．

(3) シンク・パターンが見つかると ID アドレス・マークを検出しにいきます．

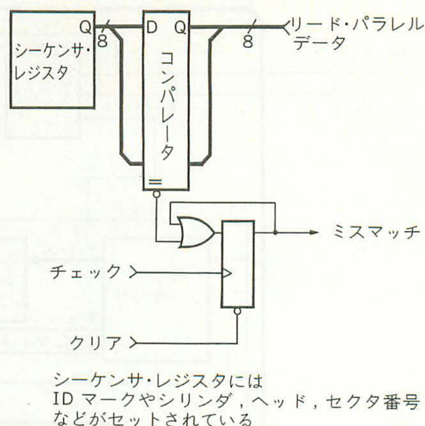
図 15 にアドレス・マーク検出回路を示します．

シンク・データの後，初めて 1 が出るのはアドレス・マークなのでこのビットがシフト・レジスタの Q_H にくるタイミングでバイト・ロック信号を作り，以後このバイト同期によってリード・データを直並列変換します．アドレス・マークが検出されたかどうかはマイクロ・プログラム・シーケンサでテストされ，エラーなら (1) へ戻ってつぎのセクタを見つけようとします．

〈図 15〉アドレス・マーク検出(シリアル-パラレル変換)



〈図 16〉セクタ・データ・テスト

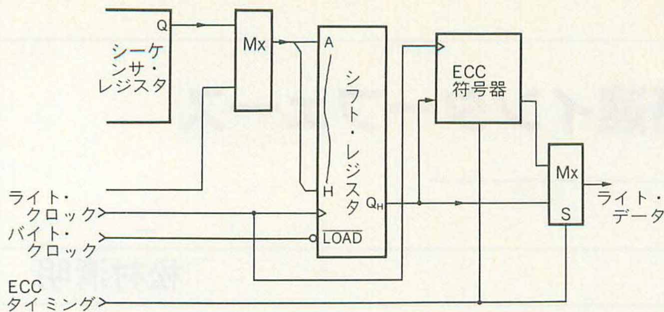


(4) マイクロ・プログラム・シーケンサはバイト・クロックごとにIDブロック内の各データを目的とするセクタのデータと一致するかテストしていきます．

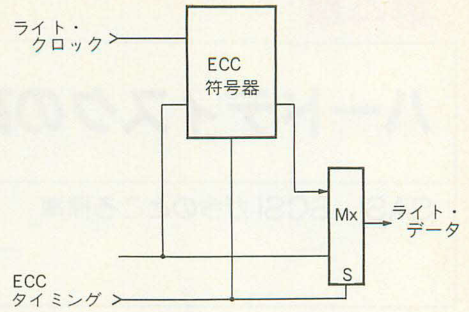
(5) CRCブロックをテストし，エラーであれば(1)へ戻ってつぎのセクタを見つけようとします．もし正常で，シリンダもしくはヘッドが異なっているときにはトラック・ミスマッチで異常終了です．セクタのみ異なっていれば(1)へ戻ってつぎのセクタを見つけようとします(図16)．

(6) 目的のセクタであれば，リード・ゲートをOFFし，ヘッド・スキャッタ時間が経過した後ヌル・データでライト・ゲートをONします．PLLシンク長分ヌル・データを書きます．

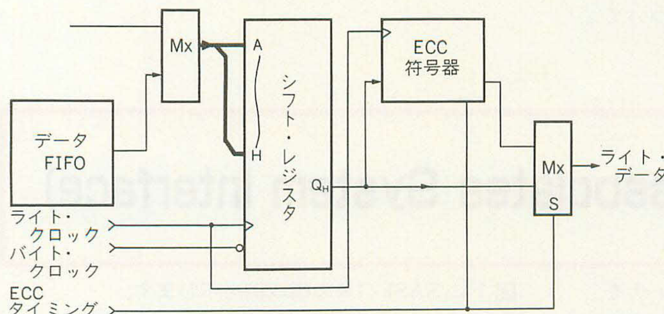
(7) シーケンサ・レジスタの内のデータ・アドレス・マークをシフト・レジスタにロードし，アドレス・マ



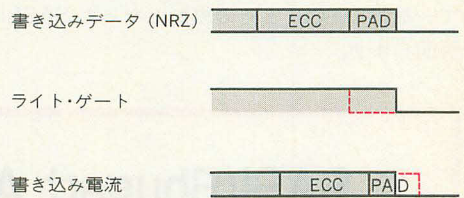
〈図17〉データ・アドレス・マーク・ライト



〈図19〉ECC ライト



〈図18〉データ・ライト



〈図20〉パッド・データ

ークを出力します．また，ECC 符号器はこの時点から ECC を生成しはじめます(図17)．

(8) データ FIFO の内容をシフト・レジスタを通して順次出力します．ECC 符号器はこの間 ECC を生成し続けます(図18)．

(9) データが終了すると ECC 符号器はレジスタの内容を出力し，ECC データを書きます(図19)．

(10) パッド・データを書き，ライト・ゲートを OFF します．

図20に示すように HDC が ECC データを出力した後，ただちにライト・ゲートを OFF すると NRZ データからヘッドの書き込み電流になるまでの遅れ(遅れについては図2を参照)によって ECC データの途中でライトが中断されてしまいます．そこで，ECC データが最後まで書かれるよう余分にライトされるデータをパッド・データといいます．

以上でセクタのライトは終了です．リードにおいてはデータ部に対しても ID 部と同様にステップ(2)から(5)を実行します．データが FIFO を通して HDC から外へ読み出されることと CRC チェックのかわりに ECC チェックが行われることが異なるだけです．

また，ここでは DMA で HDC とメモリの間のデータがやりとりされることを前提として，データ FIFO があるものとして説明しました．しかし，HDC の内部もしくは外部に専用メモリを持つタイプの HDC の場合にはデータ FIFO がないものもあります．

ハードディスク・コントローラはこのようなフォーマット処理を内部で実行するわけですが，時間依存性がきわめて高い処理であり，HDC をコントロールするファームウェアもこの時間依存を考えてチューニングするにはかなりの困難がともないます．

また，記録方式やシーク系のコントロールも年々変化し，方法論の変化も手伝ってメガバイトあたりの単価も急速に低下しています．

ハードディスク・メーカ以外の者が HDC を直接コントロールするようなハードウェアおよびソフトウェアを作ることは技術的にもコスト的にもほとんど意味がなくなってきたといえるでしょう．

そこで，次章で述べる SCSI インターフェースを通じてハードディスクを使用することがますます重要になっていくと思われます．

◈参考・引用*文献◈

(1)*モハメッド・マニア他：ファイア符号用チップを用いてディスクのデータ誤りを訂正，日経エレクトロニクス，1981．8．17，pp.155-162，日経マグロウヒル．

(2) 原島博他：誤り訂正符号化技術の応用事例，トリケップス，1986．

(3) NEC，μPD7261　HDC ユーザーズマニュアル，1983．

(4) NEC，μPD7263　HDC ユーザーズマニュアル，1986．

(5) NEC，CPD データブック(μPD72061，μPD7262 他)，1989．

(6) 日立，HD63463 ユーザーズマニュアルズ，1985．

(7) 富士通，ハードディスクコントローラ MB89342A ユーザーズマニュアル，1989．

(8) Adaptec, inc. AIC-270 Disk Controllers, 1986．

(9) Western Digital, WD5010-10, Winchester Controller, 1986．

第4章

ハードディスクの高速インターフェース

SASI，SCSI が今のところ標準

松村清明

パーソナル・コンピュータ用インターフェースとして一般的なSCSIとその原形となったSASIについて説明します．

① SASI(Shugart Associates System Interface)

SASIはシュガート社がマイクロプロセッサをCPUとするような小規模コンピュータの周辺機器用インターフェースとして考案しました．これは主に磁気ディスク装置のインターフェースを目的としたものです．

当時小型ハードディスク・インターフェースはフロッピ・ライク・インターフェースといわれるST506インターフェースが主流でした．しかし，データ・レートがフロッピ・ディスクの10倍(5Mbps)であるため，インターフェースを簡単に作ることができませんでした．

そこで，タイミング的に自由度が高く10～20本程度のTTLで簡単にインターフェースできるSASIが作られ，マイクロコンピュータ・システムやパーソナル・コンピュータにハードディスクを接続するために使用されるようになりました．

SASIバスのシステム構成

ホスト(SCSIではイニシエータ)の数は1台に固定でアドレスはもちません．コントローラ(SCSIではターゲット)は複数台接続することができます．

図1にSASIバスの構成例を示します．

セレクション・フェーズ(SCSIバスの状態遷移の項を参照)で，SASIバスのデータ・バスの$-DB_0$がアクティブなときに選択されるターゲットがID=0，$-DB_7$がアクティブなときに選択されるターゲットがID=7として最大8台まで同一バスに接続できます．

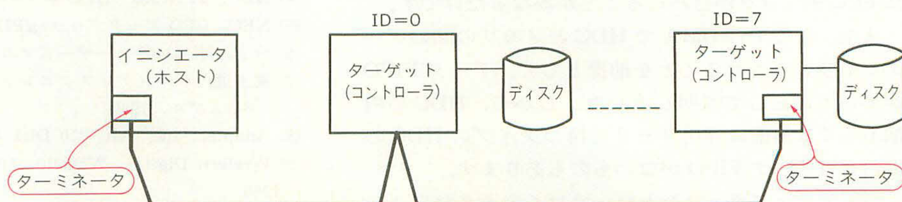
ターミネータは両端の機器が固定的に持ち，機器の電源によりターミネータ・パワーが供給されます．SASIバスにはターミネータにバスから電力を供給する考え方はありません．

以下の説明ではSCSIとの表現の混同をさけるためホストとコントローラをSCSIの呼び方であるイニシエータとターゲットに統一して説明します．

SASIバス上のすべての信号は，SCSIのシングル・エンド型と同様，負論理("L"でアクティブ)です．各信号の詳細についてはSASIバス信号の項を参照してください．

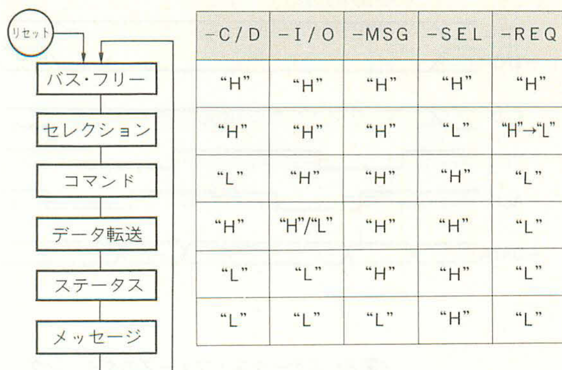
SASIバスの状態遷移

SASIバスの遷移状態は後に述べるSCSIバスと比較すると単純で，図2のように常にバス・フリー，セ



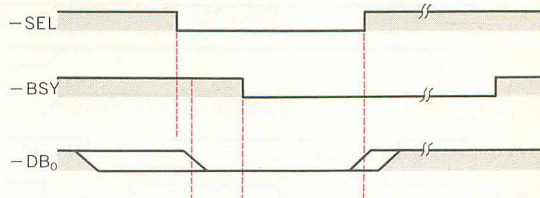
〈図1〉
SASIバスの接続例

〈図 2〉バスの状態遷移

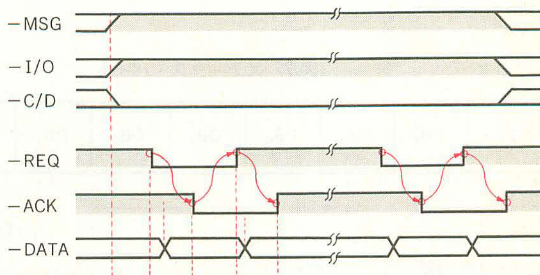


−C/D	−I/O	−MSG	−SEL	−REQ
“H”	“H”	“H”	“H”	“H”
“H”	“H”	“H”	“L”	“H”→“L”
“L”	“H”	“H”	“H”	“L”
“H”	“H”/“L”	“H”	“H”	“L”
“L”	“L”	“H”	“H”	“L”
“L”	“L”	“L”	“H”	“L”

〈図 3〉セレクション・フェーズのタイミング



〈図 4〉コマンド・フェーズのタイミング



〈表 1〉コントロール・ラインとフェーズ

信号				フェーズ名	データ転送
-SEL	-MSG	-C/D	-I/O		
“H”	“H”	“H”	“H”	データ・アウト	イニシエータからターゲットへ
“H”	“H”	“H”	“L”	データ・イン	ターゲットからイニシエータへ
“H”	“H”	“L”	“H”	コマンド	イニシエータからターゲットへ
“H”	“H”	“L”	“L”	ステータス	ターゲットからイニシエータへ
“H”	“L”	“L”	“L”	メッセージ	ターゲットからイニシエータへ
“L”	“H”	“H”	“H”	セレクション	イニシエータからターゲットへ
“H”	“H”	“H”	“H”	バス・フリー	

レクション，コマンド，データ転送(コマンドの種類により，ない場合がある)，ステータス，メッセージ，バス・フリーの各フェーズの順で動作します．

図 2 に SASI バスの状態遷移を示します．

現在のフェーズはコントロール・ラインによって決定されます．コントロール・ラインとフェーズの関係を表 1 に示します．−SEL はイニシエータがドライブするライン，−MSG，−C/D，−I/O はターゲットがドライブするラインです．いずれのラインもオープン・コレクタでドライブされる負論理の信号です．

イニシエータはひとつと決まっているので SCSI でのアービトレーション・フェーズのようなターゲット，イニシエータ間のバス専有のためのフェーズはまったくありません(アービトレーション・フェーズについては，SCSI の項で説明)．

(1) バス・フリー・フェーズ

バス上のすべての機器がバスをドライブしていない状態で，−BSY が “H”(インアクティブ)です．

(2) セレクション・フェーズ

イニシエータがターゲットを選択します．

イニシエータはバス・フリー・フェーズであることを確認してデータ・ラインにターゲットの ID を出力し，−SEL をアクティブにすることによってターゲットを選択します．選択されたターゲットは−BSY をアクティブにして選択されたことを示します．このときのタイミングを図 3 に示します．

(3) コマンド・フェーズ

イニシエータからターゲットにコマンドが送出されます．−C/D＝“L”，−I/O＝“H”，−MSG＝“H”でコマンド送出の状態で−REQ がアクティブ(“L”)となります．ここで−REQ，−ACK のハンドシェイクでイニシエータよりターゲットにコマンドが送られます．

コマンドの長さは 6 バイト固定です．このときのタイミングを図 4 に示します．

(4) データ転送フェーズ

イニシエータからターゲット，またはターゲットからイニシエータへデータが転送されます．−MSG＝“H”，−C/D＝ “H” となると，データ転送モードとなります．−I/O が “H” のときイニシエータからターゲット，“L” のときターゲットからイニシエータへデータ転送されます．このときのタイミングを図 5 および図 6 に示します．

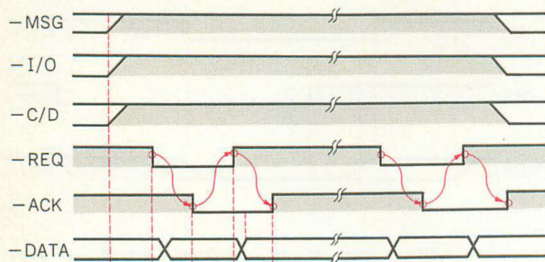
(5) ステータス・フェーズ

ターゲットからイニシエータへステータス・バイトを送出します．ステータス・バイトの形式を図 7 に示します．

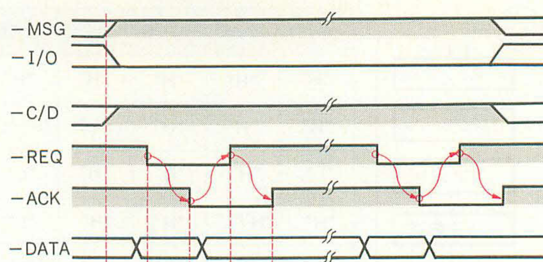
DB_0がパリティ・エラー，DB_1はコマンド実行中のエラー，DB_2～DB_4は使用しません．通常 0 です．DB_5～DB_7は論理ユニット番号(SASI コマンドの項を参照)に対応しています．

−C/D＝ “L”，−I/O＝ “L”，−MSG＝ “H” で

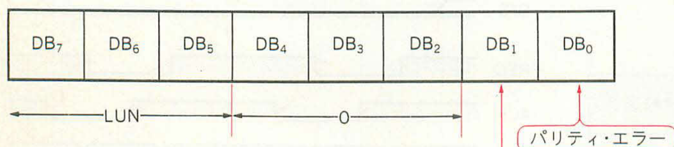
〈図5〉データ転送フェーズ(イニシエータからターゲット)



〈図6〉データ転送フェーズ(ターゲットからイニシエータ)

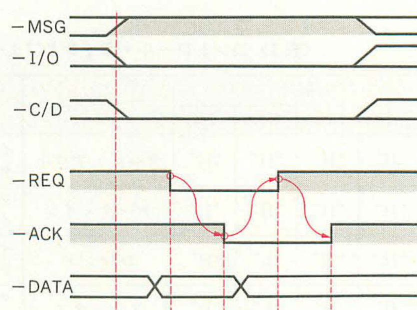


〈図7〉ステータス・バイト



LUN は論理ユニット番号.
実行エラーはコマンド実行におけるエラー.
パリティ・エラーはデータ転送中のパリティ・エラー.

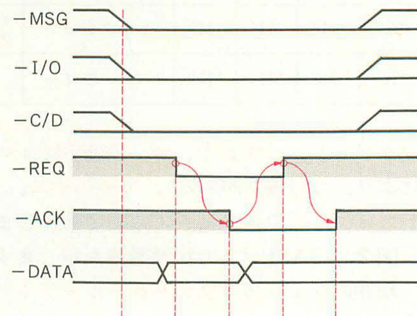
〈図8〉ステータス・フェーズのタイミング



〈図10〉コマンド・ディスクリプタ・ブロック

バイト＼ビット	7	6	5	4	3	2	1	0
0	コマンド・グループ			オペレーション・コード				
1	論理ユニット番号(LUN)			論理ブロック・アドレス2(MSB)				
2	論理ブロック・アドレス1							
3	論理ブロック・アドレス0 (LSB)							
4	転送長							
5	コントロール・バイト							

〈図9〉メッセージ・フェーズのタイミング



ステータス・バイトを送出します．このときのタイミングを図8に示します．

(6) メッセージ・フェーズ

ターゲットからイニシエータにメッセージ・バイトとして0(SCSIバス規格におけるコマンド・コンプリート・メッセージに当たる)を送出します．−C/D＝“L”，−I/O＝“L”，−MSG＝“L”でメッセージ・バイトを送出します．このときのタイミングを図9に示します．

(7) バス・フリー

メッセージ・フェーズを終了するとターゲットは−BSYを“H”にしてバスを開放します．

SASIコマンド

SASIコマンドは6バイトのコマンド・ディスクリプタ・ブロック(CDB)で構成されます．

コマンド・ディスクリプタ・ブロックの一般形式を図10に示します．

0バイト目のDB_7〜DB_5でコマンド・グループを表し，通常のディスク・コマンドはグループ0として規定されています．これに対し機器ごとの特殊コマンドはグループ7もしくは6をベンダごとに独自に使用しています．

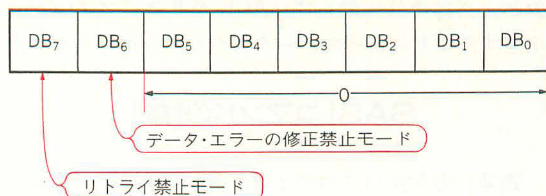
オペレーション・コードはグループ内でのコマンドのナンバを示します．

論理ユニット番号(LUN)はひとつのコントローラで複数のドライブをコントロールする場合のドライブ番号です．

論理アドレス2，1，0は21ビットのレコード・アドレスで，ディスクに対してはセクタ0，ヘッド0，シリンダ0から始まるセクタ・アドレスです．論理アドレスの計算は，

論理アドレス＝(シリンダ・アドレス×シリンダ当たりのヘッド数＋ヘッド・アドレス)×トラック当たりのセクタ数＋セクタ・アドレス

〈図 11〉 コントロール・バイト



となります．

転送長はひとつのコマンドでアクセスする連続したセクタの数で，1は1セクタ，FFは255セクタを示し，0は256セクタとなります．

コントロール・バイトは図11のようにふたつのコントロール・ビットをもっています．DB_7がリトライの禁止モードで，1の場合リトライを禁止し，0の場合リトライを許可します．DB_6は修正可能データ・エラーに対してデータの修正をすることを禁止するモードです．1の場合データの修正を禁止し，0の場合データの修正を許可します．

SASIバス信号

SASIバスは50ピンのコネクタにより接続され，すべての信号線はGND線とペアで構成されています．信号はすべて負論理で，双方向のデータ・ラインと単方向のコントロール・ラインよりなります(図12)．

以下にバス信号の意味について述べます．

(1) $-DB_0$～$-DB_7$，$-DB_P$(データ・バス)

データは−I/Oが“L”でターゲットからイニシエータ，−I/Oが“H”でイニシエータからターゲットの方向にドライブされます．また−SELがアクティブなときにイニシエータがドライブし，ターゲットは8ビットの内の1ビットと−SELのANDをとってセレクトされたことを認識します．

(2) −BSY(ビジー)

ターゲットがドライブします．−SEL(イニシエータがドライブする)によって選択されたターゲットがドライブ(“L”)します．ターゲットがほかのコントロール・ラインをドライブするためには，まず−SELによって選択されなければなりません．このラインをドライブしている間，ターゲットはバスを専有しています．コマンドの終了によってターゲットはこのラインを解放します(すなわちバスを解放)．

(3) −SEL(セレクト)

イニシエータがドライブします．イニシエータがターゲットを選択するために用います．選択されるとターゲットは−BSYをアクティブ(“L”)にします．イニシエータはバスがフリーである(−BSYが“H”)ことを確認し，このラインをドライブ(“L”)することで目的のターゲットを選択できます．

〈図 12〉
SASI バス

ピン	信号名	イニシエータ/ターゲット
2	$-DB_0$	←→
4	$-DB_1$	←→
6	$-DB_2$	←→
8	$-DB_3$	←→
10	$-DB_4$	←→
12	$-DB_5$	←→
14	$-DB_6$	←→
16	$-DB_7$	←→
18	$-DB_P$	←→
36	−BSY	←
38	−ACK	→
40	−RST	→
42	−MSG	←
44	−SEL	→
46	−C/D	←
48	−REQ	←
50	−I/O	←

ただし，これらのピンと対になる奇数ピンはすべてGND．また図中の矢印は信号の方向

(4) −RST(リセット)

イニシエータがドライブします．

最小400ns，最大10sのパルスを出力することにより，バスおよびバスに接続されているターゲットを初期化するためにイニシエータがドライブします．なお，電源投入時には本信号を用いてターゲットをイニシャライズすることが望ましいとされています．

これによりバスはフリー(−BSYが“H”で，イニシエータもしくはすべてのターゲットがラインをドライブしていない状態)であり，イニシエータは任意のターゲットを選択できます．

(5) −I/O(インプット/アウトプット)

データ・ラインのドライブ方向を決定する信号でターゲットがドライブします．

“L”レベルでターゲットがデータ・ラインをドライブし，イニシエータがデータを受け取ることを示します．

“H”レベルでイニシエータがデータ・ラインをドライブし，ターゲットがデータを受け取ることを示します．

(6) −C/D(コマンド/データ)

ターゲットがドライブします．

データ・ライン上のデータがコマンドかデータかを示します．

“H”レベルでデータ・フェーズであることを示します．

(7) −MSG(メッセージ)

ターゲットがドライブします．

“L”レベルでメッセージであること，すなわちコマンドの実行完了を示します．

(8) −REQ(リクエスト)

ターゲットがドライブします．

−ACK(イニシエータがドライブする)と対で用いられ，データのハンドシェイクを行います．−I/O，−C/D，−MSG との関係により，転送モードが規定されます．

(9) −ACK(アクノリッジ)

イニシエータがドライブします．

−REQ(ターゲットがドライブする)と対になり，データのハンドシェイクに用いられます．

ターゲットが−REQ をドライブ("L")するとイニシエータはそれに対して−ACK をドライブ("L")し，応答することによってデータのやりとりが行えます．

SASI コマンドの詳細

表 2 に SASI バスのディスク・コマンドの一部を示します．

ディスク・アクセスのコマンドとして共通で使用されていたのはこれらのグループ 0 のコマンドのみで，

SASI インターフェース回路

簡単な SASI コントローラ(イニシエータ)の回路例を図 A に示します．

アービトレーション・フェーズがある SCSI と異なり，SASI はタイミングが単純です．そこで，マイクロコンピュータと接続する場合には図のような簡単な回路で SASI ディスクなどと接続することができます．

(1) データ・バッファ

入力時には−REQ がアクティブならばデータは確定しているので，直接 SASI バス上のデータをインプットし，インプットの終わりで−ACK を発生させます．

出力時には−REQ がアクティブな状態でアウトプットし，ラッチしてアウトプットの終わりで−ACK を発生させます．

セレクト時にはセレクトする ID をあらかじめアウトプットしておき，−SEL をアクティブにすることでターゲットをセレクトします．

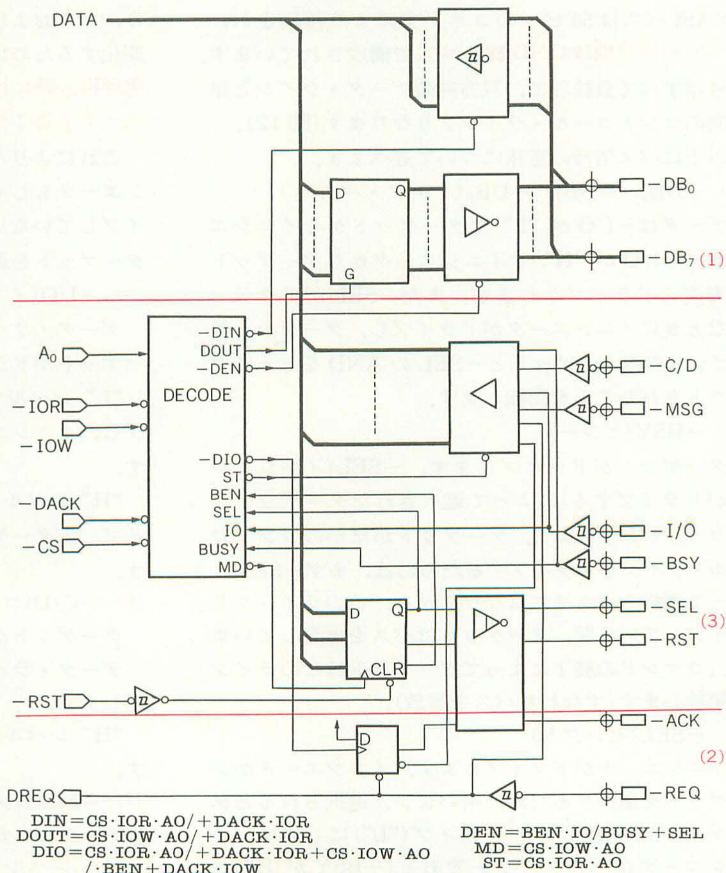
(2) REQ/ACK ハンドシェイク

−REQ がアクティブな状態でデータをインプットもしくはアウトプットすると−ACK をアクティブにし，−REQ がインアクティブになるまで−ACK の出力を続けます．

(3) ステータス，コントロール

−SEL はターゲットをセレクトするためのラインです．データ・バッファ ID をあらかじめセットしておき，このラインをアクティブにします．−RST は SASI バスのリセット・ラインです．−C/D，−MSG，−I/O および−BSY はターゲットからのフェーズを表す信号で，これにもとづいてバスのドライブをコントロールしたり，プログラムによってデータの流れをコントロールします．DECODE は各 IC をコントロールする信号を発生するための PLD です．

〈図 A〉 簡単な SASI インターフェース



〈表 2〉SASI バスのディスク・コマンド

00H	テスト・ユニット・レディ
01H	リゼロ・ユニット
03H	リクエスト・センス
08H	リード
0AH	ライト

〈図 13〉テスト・ユニット・レディ

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 14〉リゼロ・ユニット

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 15〉リクエスト・センス

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

それ以外で必要なコマンドはコントローラのメーカによってグループ 6，7 などに独自に決められていました．

グループ 0 のコマンドとしてこのほかにフォーマット・コマンドも使用されていましたが，代替処理を行わないものが多く，実際的にはコントローラによりフォーマットの方法はまちまちでした．

これらのコマンドの詳細は SCSI の項で述べます．

(1) テスト・ユニット・レディ(00H)

この命令はターゲットがレディ状態であるかどうかを調べるための命令です．命令に対してターゲットはリード/ライト動作を含むすべての命令の実行が可能なとき，正常終了(ステータス 00H)し，その他のときにはチェック・コンディション(ステータス 02H)となります(図 13)．

〈図 16〉センス・データ(4 バイト)

バイト＼ビット	7	6	5	4	3	2	1	0
0	センス・コード							
1	LUN			論理ブロック・アドレス 2 (MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0 (LSB)							

〈図 17〉リード

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0
1	LUN			論理ブロック・アドレス 2 (MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0 (LSB)							
4	転送長							
5	コントロール・バイト							

〈図 18〉ライト

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0
1	LUN			論理ブロック・アドレス 2 (MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0 (LSB)							
4	転送長							
5	コントロール・バイト							

(2) リゼロ・ユニット(01H)

この命令を受けたときには，装置内部のヘッド位置決め系の再初期化を行い，シリンダ 0 へシークします(図 14)．

(3) リクエスト・センス(03H)

この命令はターゲットがチェック・コンディション・ステータスを送出した際にイニシエータが詳細情報を読み取るための命令です(図 15)．

ターゲットは 4 バイトのセンス・データ(図 16)を送出します．SCSI コマンドの非拡張センス・データがこれと同じです．詳しくは SCSI コマンドのリクエスト・センスの項で述べます．

(4) リード(08H)

この命令はイニシエータがターゲットに対して磁気ディスク媒体に記録されているデータの転送を要求する命令です．コマンド・ディスクリプタ・ブロックの論理ブロック・アドレスはデータ転送を要求する先頭のブロックを示し，転送長は転送を要求するブロック数を示します(図 17)．

(5) ライト(0AH)

イニシエータがターゲットに対して磁気ディスク媒体にデータの書き込みを要求する命令です(図 18)．

コマンド・ディスクリプタ・ブロックの詳細はリード・コマンドと同じです．

② SCSI(Small Computer System Interface)

SCSIはSASIを基本に，X3T9.2として1982年4月からアメリカ規格協会(ANSI)で検討が始められました．その後2カ月間隔でANSIのX3委員会により公式会議が開かれ，1986年6月にANSIにより承認されアメリカ国家規格ANSI　X3.131-1986“Small Computer System Interface”として制定されました．

SASIバスではイニシエータはひとつであったのに対しSCSIでは複数のイニシエータが1本のSCSIバス上にあることを認めたためアービトレーション(調停)が必要になり，イニシエータもアドレスをもつことになりました．SASIでは一度コネクトが確立されるとコマンド終了までバスを占有していました．これに対してSCSIでは複数のイニシエータがバスを使用し，マルチプロセスOSにおいては同時に複数のターゲットをアクセスするためバスがフリーになるのを待つことになるような場合が起こります．

そこでコマンド起動後データ転送がないときには一度ディスコネクトしてバスを開放し，再度リセレクションすることによりバスの占有時間を短縮し，バス使用効率を上げることを可能にしました(このような例をSCSIバスの状態遷移の項で述べる)．ただし，同時にひとつのアクセスしか発生しない場合にはディスコネクトしないほうが高いスループットを得ることができます．

SASIではコネクトが確定するとバス・リセットを除いてディスコネクトまでターゲットがすべてのシーケンスをコントロールしていました．SCSIでは－ATN(アテンション)によりメッセージ・システムを通してイニシエータからターゲットに指示ができるようになりました．

SCSIバスのシステム構成

SCSIバスには8台の装置が接続でき，それぞれが0～7のID番号をもちます．ID＝7の優先順位がもっとも高く，ID＝0の優先順位が最低です．それぞれの役割もSASIのように固定的でなく，イニシエータ(命令を出す側)にもターゲット(命令を実行する側)にもなることができます．ただしこれはハードウェアの機能によります．

図19にSASIバスの接続例に相当するSCSIバスの接続例を示します．これはイニシエータ1台に対して1～7台のターゲットが接続されている例です．

ターゲットはSASIと同様にID＝0から順にIDを割り当てられているのが一般的です．またイニシエータは逆にID＝7から順にIDを割り当てることが多いようです．ただし重複しない任意のIDを任意の順で用いてもかまいません．

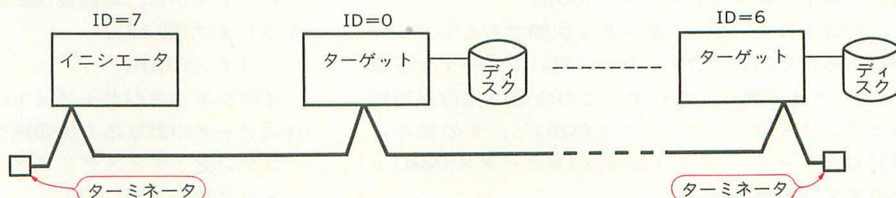
ターミネータは両端機器の外部にもち，バスのTERMPWRによりターミネータ・パワーが供給されます．各信号の詳細についてはSCSIバス信号の項を参照してください．またSASIバスと同様に負論理(“L”がアクティブ)のシングル・エンド型を例として以下で説明します．

SCSIバス状態遷移

SCSIバスのもっとも単純な状態遷移は図2で示したSASIの状態遷移と同じであり，SASIバスに対して上位方向の互換性をもっています．これに対してアービトレーションを行う，より一般的なSCSIバスの状態遷移を図20に示します．またANSIでは状態遷移中の時間を厳密に規定しています．この時間規定を表3に示します．以下，各状態の遷移について順に説明していきます．

(1) いかなるフェーズにいてもリセットによってバス・フリー状態になります．またバスを占有しているターゲットはコマンド，データ，ステータス，メッセージの各フェーズからハンドシェイクの単位ごとの任意のタイミングでディスコネクトしてバス・フリー・フェーズへ戻ることができます．

(2) バスを使用したいイニシエータもしくはターゲットはバス・フリーであることを認識(－BSY＝



〈図19〉
SCSIバスの接続例

"H")してアービトレーション・フェーズに入ります.

図21でアービトレーションからセレクションまでの状態遷移を説明します.

SCSIデバイス(ID_5)は，バス・フリー・フェーズとなるのを待ち，－BSY信号がアクティブであることを検出後バス・フリー時間を待ってSCSIバス・フリー状態終了後からバス・セット時間以内に－DB_5信号とその－BSY信号をアクティブ("L")にします(図21ではバス・セット時間は0に等しい).

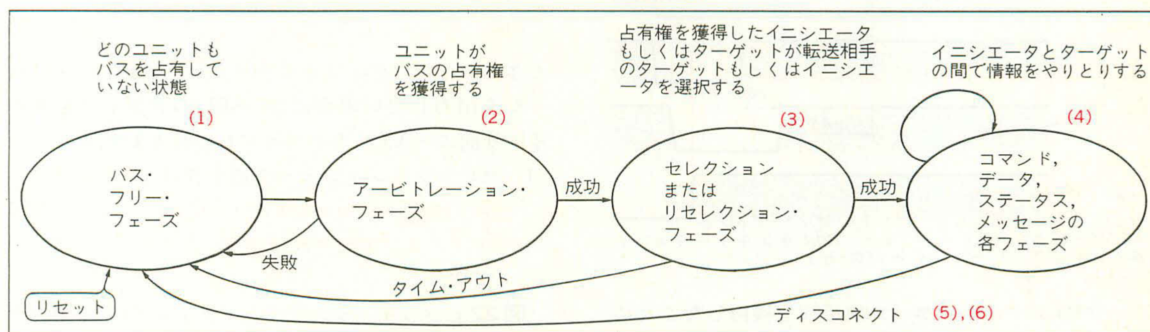
つぎにアービトレーション・ディレイだけほかのDB信号を監視します.

SCSIデバイス(ID_7)は同時刻(ID_5より若干遅れた時間)にバス・フリー・フェーズを検出します．ID_7のデバイスはバス・フリー時間，バス・セット時間をID_5のデバイスと同じように待ったのち，－DB_7信号とその－BSYをアクティブ("L")にします．つぎにアービトレーション・ディレイだけID_5のデバイスと同じようにほかのDB信号を監視します.

ここでID_7のデバイスがID_5のデバイスより優先となるのでID_5のデバイスは－DB_7信号がアクティブ("L")となっていることを検出すると－BSYと－DB_5のドライブをバス・クリア・ディレイ以内に中止します.

ID_7のデバイスは引き続きアービトレーション・ディレイだけ待ったのち－SEL信号をドライブします.

〈図20〉 SCSIバスの状態遷移



〈表3〉 SCSIバスの時間規定

(1)アービトレーション・ディレイ(最小2.2μs)
アービトレーション・フェーズでSCSI装置が－BSYをアクティブ("L")にしIDをデータ・バスに送ってから優先順位が確定するまでの最小時間

(2)アサーション時間(最小90ns)
同期データ転送において－REQもしくは－ACKの最小パルス幅

(3)バス・クリア・ディレイ(最大800ns)
次の三つの要因が発生後，SCSI装置が信号のドライブを終了するまでの最大時間
a. バス・フリー・フェーズの検出の場合
b. アービトレーション・フェーズ中，他のデバイスからSEL信号を受けた(アービトレーション・フェイル)
c. －RST信号がアクティブ("L")になった

(4)バス・フリー・ディレイ (最小800ns)
アービトレーション・フェーズを開始するとき，バス・フリー・フェーズ検出から－BSYとIDをドライブするまでの最小待ち時間

(5)バス・セット・ディレイ(最大1.8μs)
アービトレーション・フェーズを始めるとき，バス・フリー・フェーズ検出から－BSYとIDをドライブするまでの最大許容時間

(6)バス・セトル・ディレイ(最小400ns)
信号状態変化後，バスの安定を待つべき最小時間

(7)ケーブル・スキュー・ディレイ(最大10ns)
任意の二つのバス信号線の間の伝搬時間の最大差

(8)データ・リリース・ディレイ(最大400ns)
－I/O信号が"H"から"L"に変化後，イニシエータがデータ・バスのドライブを終了するまでの最大時間

(9)デスキュー・ディレイ(最小45ns)
バス信号間の遅れの差を補正するために必要な最小時間

(10)ホールド・タイム(最小45ns)
同期データ転送で，イニシエータまたはターゲットが－ACKまたは－REQをドライブしてからデータ・バス上のデータを保持しておくべき最小時間

(11)ネゲーション時間(最小90ns)
同期データ転送で，REQ信号から次のREQ信号またはACK信号から次のACK信号までのインアクティブ状態の最小時間

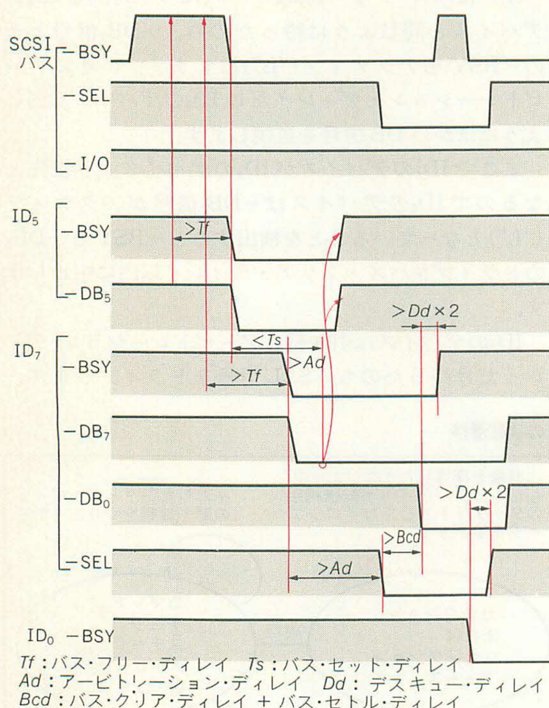
(12)リセット・ホールド・タイム(最小25μs)
－RST信号をアクティブ状態に保持しておく最小時間

(13)セレクション・アボート・タイム(最大200μs)
セレクション，リセレクション・フェーズでSCSI装置が選択されたことを認識後，－BSYで応答するまでの最大許容時間

(14)セレクション・タイム・アウト・ディレイ(最小250ms推奨)
セレクション，リセレクション・フェーズでタイム・アウト処理を始めるまでにSCSI装置からのBSY信号を待つべき最小時間．実際にはこれよりかなり長い時間(たとえば数秒)待つことが多い

(15)トランスファ時間(180ns～1020ns)
同期データ転送中でのREQ信号，ACK信号の最小繰り返し時間・メッセージ・フェーズによりイニシエータ・ターゲット間で個別に規定された時間(メッセージ・システムの項を参照)

〈図 21〉 SCSI バスの状態遷移
(アービトレーション〜セレクション・フェーズ)



これでID_7のデバイスがバス使用権を獲得したことになります．図 21 の前半がこれに当たります．

(3) イニシエータならセレクション・フェーズ，またターゲットならリセレクション・フェーズに入ります．

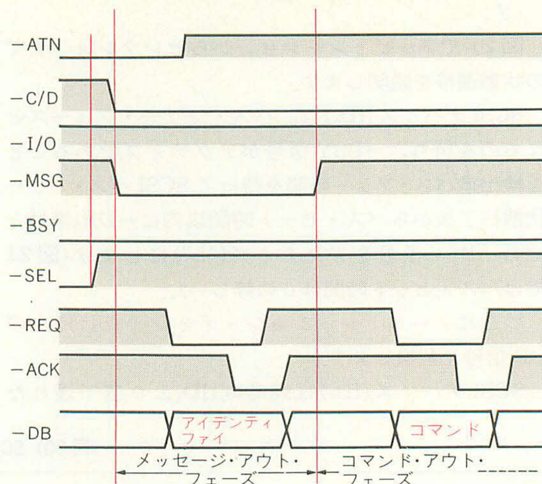
セレクション・フェーズは図 21 の後半，リセレクション・フェーズは図 23 の最後で示しています．セレクション・フェーズとリセレクション・フェーズはセレクトされた時点で－I/O が“H”か“L”かで決まり，セレクションでは“H”，リセレクションでは“L”です．

ここでは図 21 にそってID_7がイニシエータでID_0のターゲットをセレクトする場合について説明します．－SEL をドライブした後，バス・クリア・ディレイ＋バス・セトル・ディレイ待ってセレクトしたいデバイスの ID(この場合ID_0)に対応する－DB_0をドライブします．

つぎに，デスキュー・ディレイの二倍待って－BSY のドライブをやめ，ターゲットが－BSY をドライブするのを待ちます．ターゲットの応答がなければタイム・アウトとしてデータ・バスのドライブをやめ，つぎに－SEL のドライブをやめてバス・フリー・フェーズへ戻ります．

ターゲットからの応答があればデス・キュー・ディレイの二倍待って－SEL のドライブとデータ・バスのドライブを終了し，つぎのフェーズへ移行します．この

〈図 22〉 SCSI バスの状態遷移(コマンド・フェーズ)



とき，アイデンティファイ・メッセージなどのメッセージを出力したい場合には－SEL のドライブをやめるより前に－ATN をドライブしておきます．

(4) セレクション・フェーズのつぎは－ATN がアクティブならメッセージ・アウト・フェーズ，－ATN がアクティブでなければコマンド・アウト・フェーズです．

図 22 にメッセージ・アウト・フェーズ(アイデンティファイ・メッセージ)を含む場合のコマンド・アウト・フェーズの開始部分のタイミングを示します．アイデンティファイ・メッセージについてはメッセージ・システムの項を参照してください．

ターゲットはセレクション・フェーズの終わりで－ATN がアクティブである間はメッセージ・アウト・フェーズと必要ならその応答のフェーズを続けます．

ここでイニシエータはアイデンティファイ・メッセージなど(詳しくはメッセージ・システムの項を参照)のメッセージを出力します．

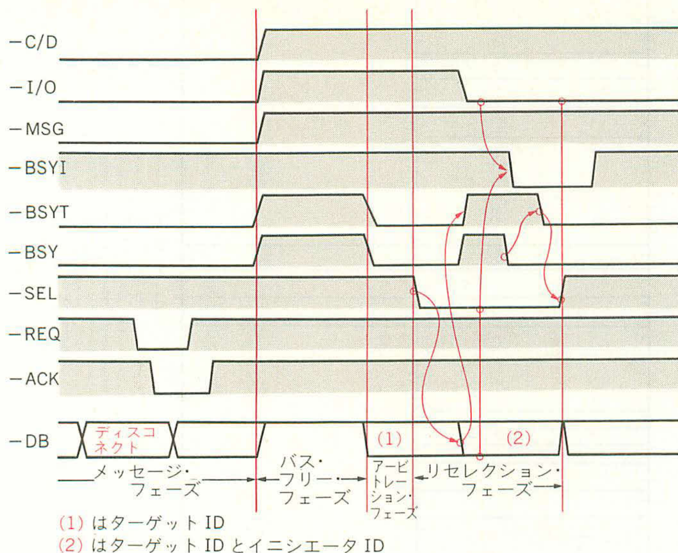
次にコマンド・フェーズに入り，6 バイトから 12 バイトのコマンドをイニシエータがターゲットに出力します．コマンドのバイト数と内容は SCSI コマンドの項を参照してください．

(5) アイデンティファイ・メッセージでディスコネクトできることをイニシエータがターゲットに示した場合，ディスコネクトできるターゲットはコマンドの終了までの間，もしくはデータ転送の準備ができるまでの間，ディスコネクトしてバスを開放します．

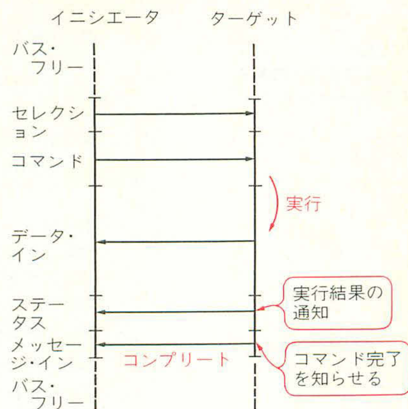
図 23 にバスの開放とその後のターゲットによるイニシエータのリセレクション・フェーズを示します．

通常ターゲットはディスコネクトする前にセーブ・データ・ポインタなどとともにディスコネクト・メッセージを出力し，つぎにディスコネクトします(メッセ

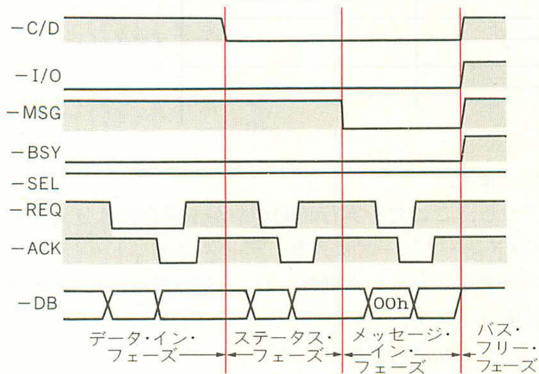
〈図 23〉 SCSI バスの状態遷移(ディスコネクト, リセレクション・フェーズ)



〈図 25〉 単純な SCSI シーケンス



〈図 24〉 SCSI バスの状態遷移(データ転送, 終了フェーズ)



ージ・システムの項を参照). データ転送やコマンド終了の準備が整うとターゲットはイニシエータをリセレクトします. ターゲットはアービトレーション・フェーズの後に−SEL をアクティブにし, イニシエータの ID を出力するとともに−I/O をアクティブにしてこれがリセレクションであることを示します.

つぎに一度−BSY のドライブをやめ, イニシエータが応答して−BSY をドライブしてくるのを待ちます. イニシエータが応答してくるとターゲットはふたたび−BSY をドライブして−SEL のドライブを終了し, つぎのフェーズへ移行します. この間−I/O はドライブしたままです.

(6) 最後にコマンド終了とそれにともうバス・フリー・フェーズへの移行を図 24 に示します.

この例ではディスク・リードやセンス・ステータスなどのデータ・イン・フェーズに続けて終了フェーズへ移行する場合を示しています. データ転送フェーズが終了すると SASI バスの状態遷移と同様にステータス・フェーズに入ります. ここでターゲットは 0(正常終了)もしくは 2(エラー, 例外状態, 異常状態での終了)などのステータスをイニシエータに出力し, つぎにメッセージ・アウト・フェーズで 0(コマンド終了)を出力したのちディスコネクトします.

以上でコマンド実行にともなう一般的な状態遷移を説明しましたが, 一台のディスクをアクセスするだけの単純なシステムでは SASI の項で説明したもっとも単純な状態遷移の場合にスルー・プットが最大になることに注意しなければいけません.

不必要に複雑なシーケンスにするとスループットはどんどん低下します.

● 簡単なシーケンス

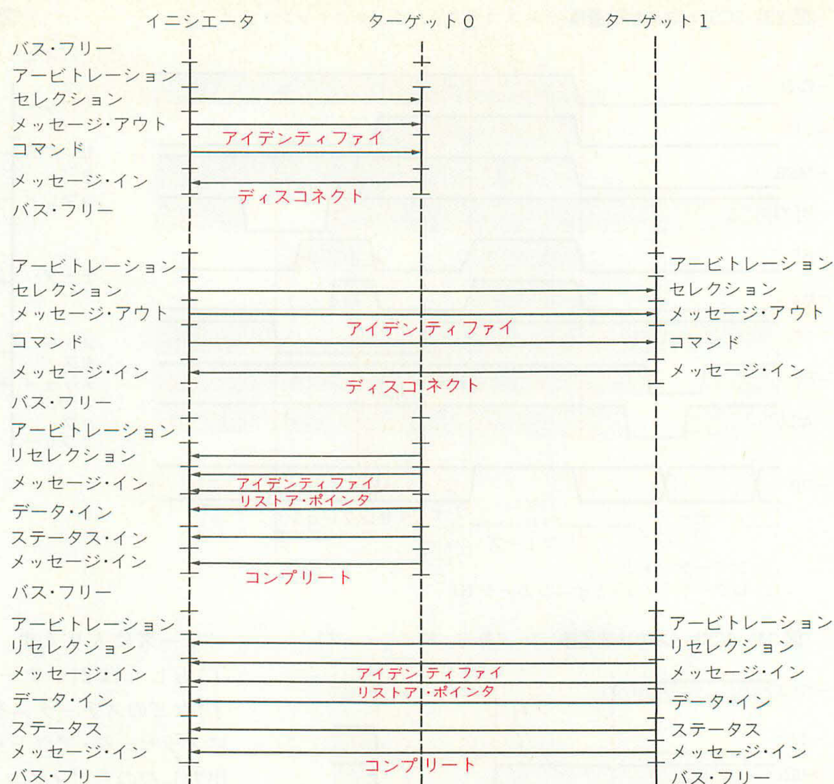
図 25 にもっとも単純な SCSI シーケンスを示します.

イニシエータはバス・フリー・フェーズであることを確認してセレクション・フェーズに入ります. セレクション・フェーズでターゲットを選んだイニシエータは, つぎにコマンド・フェーズに移行し, ターゲットはイニシエータからコマンドを得て実行します. データ・イン・フェーズではコマンド実行にともなうデータ転送を行います. つぎにターゲットはステータス・フェーズによってコマンドの実行結果をイニシエータに通知します. 最後にターゲットはメッセージ・イン・フェーズでコマンド完了を通知し, バス・フリー・フェーズに移行します.

図 26 にディスコネクトのある少し複雑な SCSI シーケンスを示します. この中で使用されるメッセージについてはメッセージ・システムの項を参照してください.

バス・フリー・フェーズを確認したイニシエータはア

〈図 26〉
ディスコネクトのある
SCSI シーケンス



ービトレーション・フェーズによってSCSIバスを占有します．セレクション・フェーズでイニシエータはターゲット0を選び，－ATNをアクティブにします．メッセージ・アウト・フェーズでイニシエータはアイデンティファイ・メッセージによってディスコネクトが可能であることをターゲット0に通知します．コマンド・フェーズにおいてターゲット0はイニシエータからコマンドを得て実行します．

ここでデータのやりとりがなく，しかも実行時間の長いコマンド(フォーマットなど)やデータ転送までに時間のかかるコマンド(リードなど)においては，SCSIバスのむだな占有を防ぐためにターゲット0はメッセージ・イン・フェーズに入ってディスコネクト・メッセージを通知し，バス・フリー・フェーズに移行します(一時的に接続を中断する)．

この間にマルチプロセスで動いているようなイニシエータが他のターゲットに対するアクセスが必要な場合にはアービトレーション・フェーズによってSCSIバスを占有します．

セレクション・フェーズでイニシエータはターゲット1を選び，－ATNをアクティブにします．メッセージ・アウト・フェーズでイニシエータはアイデンティファイ・メッセージによってディスコネクトが可能であることをターゲットに通知し，コマンド・フェーズでターゲット1はイニシエータからコマンドを得て実行します．

もし，ここでも実行時間がかかる場合にはメッセージ・イン・フェーズに入り，ディスコネクト・メッセージがターゲットから通知され，バス・フリー・フェーズに移行します．

データ転送の準備ができたターゲット0はここでアービトレーション・フェーズによってバスを占有し，リセレクション・フェーズにおいてディスコネクトしたイニシエータを選択します．ターゲットはメッセージ・イン・フェーズでアイデンティファイ・メッセージを送り，このリセレクションに対応する論理ユニットを通知し，つぎにリストア・ポインタを発行します．

ターゲットはデータ・イン・フェーズでコマンドに対するデータをイニシエータに送ります．ステータス・フェーズではターゲットからイニシエータへコマンドに対する実行結果を送ります．メッセージ・イン・フェーズではターゲットからイニシエータにコマンド完了を通知します．

ターゲット1もこの後ターゲット0と同様にリセレクションし，データ転送を行います．

このように単純なSCSIシーケンスと比べると複数のメッセージ・システムを使用するので，イニシエータ-ターゲットのオーバ・ヘッドは大きくなりますが，バスは有効に使用されているといえます．

それでは，それぞれフェーズについて説明します．

〈図 27〉6 バイト・コマンド・ディスクリプタ・ブロック

バイト＼ビット	7	6	5	4	3	2	1	0
0	コマンド・コード							
1	LUN			論理ブロック・アドレス 2 (MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0							
4	転送長							
5	コントロール・バイト							

〈図 28〉10 バイト・コマンド・ディスクリプタ・ブロック

バイト＼ビット	7	6	5	4	3	2	1	0
0	コマンド・コード							
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス 3 (MSB)							
3	論理ブロック・アドレス 2							
4	論理ブロック・アドレス 1							
5	論理ブロック・アドレス 0 (LSB)							
6	0	0	0	0	0	0	0	0
7	転送長 (MSB)							
8	転送長 (LSB)							
9	コントロール・バイト							

● コマンド

SCSI コマンドの概要を示します．あくまでも概要を示すのみですから内容については SCSI コマンドの詳細の項を参照してください．

図 27～図 29 に 6 バイト，10 バイト，12 バイト，それぞれのコマンド・ディスクリプタ・ブロック (CDB) を示します．

イニシエータからのコマンドは－MSG，－I/O 信号がインアクティブ (“H”) で，－C/D がアクティブ (“L”) な状態でターゲットからの要求で REQ/ACK プロトコルによって，イニシエータから転送されます．コマンド・ディスクリプタ・ブロック (CDB) の先頭バイトにはコマンド・コードが示され，コマンド・コードのグループによって CDB の長さが決まります．以降のバイトには論理ユニット番号，先頭ブロック・アドレス，転送要求ブロック数およびコントロール・バイトが続きます．

● ステータス

ステータス・バイトはそれぞれのコマンド実行終了時のステータス・フェーズにおいてターゲットからイニシエータへ転送されます．図 30 にステータス・バイトの形式を示し，以下に各ステータスの詳細を示します．

(1) 正常終了 (00H)

本ステータスはコマンドの実行が正常終了したことを示します．

(2) チェック・コンディション (02H)

〈図 29〉12 バイト・コマンド・ディスクリプタ・ブロック

バイト＼ビット	7	6	5	4	3	2	1	0
0	コマンド・コード							
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス 3 (MSB)							
3	論理ブロック・アドレス 2							
4	論理ブロック・アドレス 1							
5	論理ブロック・アドレス 0 (LSB)							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	転送長 (MSB)							
10	転送長 (LSB)							
11	コントロール・バイト							

〈図 30〉ステータス・バイト

バイト＼ビット	7	6	5	4	3	2	1	0
0	LUN			ステータス・バイト・コード				0

センス・データがセットされるエラー，例外状態，異常状態が発生したことを示すステータスです．このステータスがセットされた場合，イニシエータはリクエスト・センス・コマンドを発行して詳細情報を受け取る必要があります．

(3) ビジー (08H)

イニシエータから受け取った命令に対してターゲットがビジーでその命令の実行ができないことを示します．イニシエータは命令を再度実行する必要があります．

(4) リザベーション・コンフリクト (18H)

ターゲットがほかのイニシエータによってリザーブされているためコマンドの実行が不可能なことを示します．

● メッセージ・システム

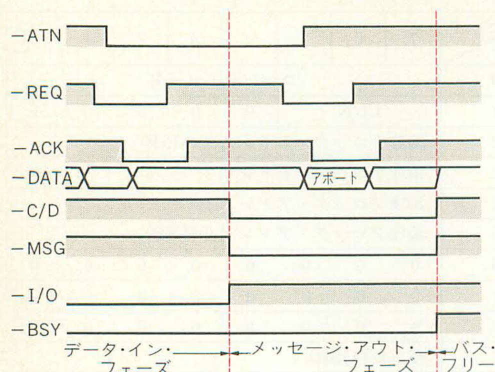
メッセージ・システムはイニシエータとターゲット間でモードをやり取りするために用いられます．

ターゲットはコマンド・コンプリート以外に数種のメッセージをサポートしています．コマンド・コンプリート以外のメッセージをサポートするイニシエータはセレクション・フェーズにおいて－SEL のドライブをやめる前に－ATN をドライブ (“L”) することでアテンション・コンディションをつくり，転送すべきメッセージがあることを知らせることができます．

この操作によってターゲットは，メッセージ・アウト・フェーズに入ります．

アービトレーション機能をもつシステムのイニシエータはアイデンティファイ・メッセージを送出してディスコネクト/リコネクト機能をもつことをターゲットに知らせることができます．この場合の状態遷移は

〈図31〉 −ATNによるメッセージの起動例



〈表4〉 メッセージ・コード

16進コード	ディスクリプション	メッセージ IN/OUT
00h	コマンド・コンプリート	IN
01h	拡張メッセージ	IN / OUT
02h	セーブ・データ・ポインタ	IN
04h	ディスコネクト	IN
05h	イニシエータ・ディテクト・エラー	OUT
06h	アボート	OUT
07h	メッセージ・リジェクト	IN / OUT
08h	ノー・オペレーション	OUT
09h	メッセージ・パリティ・エラー	OUT
0Ch	バス・デバイス・リセット	OUT
80h, C0h	アイデンティファイ	IN / OUT

SCSIバス状態遷移の項の(4)で図示されています．

またコネクト中(コネクト中のフェーズの遷移はすべてターゲットがコントロールする)にイニシエータがターゲットに対してコントロールを要求する場合，−ATNをドライブ("L")します．この要求に対してターゲットは転送の切れ目でメッセージ・アウト・フェーズに入ります．ここでイニシエータはターゲットに後述するアボート・メッセージのようなメッセージを送ることで処理を中断できます．この状態の例を図31に示します．

この例はデータ・イン・フェーズにおいてイニシエータが中断した例です．

データ・イン・フェーズにおいて−ATNをアクティブにしたのでターゲットはデータの切れ目でメッセージ・アウト・フェーズに移行しました．

ここでイニシエータはターゲットがメッセージ・アウト・フェーズに入るまでターゲットの要求する転送を続けることが重要です．

メッセージ・アウト・フェーズでアボート・メッセージを受け取ったターゲットはバス・フリー・フェーズに移行します．

表4に一般的に使用されるメッセージ・コードを示します．

(1) コマンド・コンプリート(00H)

ターゲットはコマンドの実行終了時にコマンド・コンプリート・メッセージを出力し，ディスコネクトしてバス・フリー・フェーズに戻ります．

(2) セーブ・データ・ポインタ(02H)

このメッセージはディスコネクト時にイニシエータに対して転送中のデータのポインタをセーブすることを要求するメッセージです．

実行中のコマンドのデータが全然転送されていない場合にはディスコネクト時にディスコネクト・メッセージのみを出力し，一部のデータが転送済みの場合にはセーブ・データ・ポインタ，ディスコネクト・メッセージを出力してバス・フリー・フェーズにするのが普通です．イニシエータはリセレクト時にターゲットからアイデンティファイ・メッセージを受け取ったときにセーブされているポインタをリストアする必要があります．

(3) ディスコネクト(04H)

このメッセージはターゲットがディスコネクトすることをイニシエータに知らせるためのメッセージです．出力後，ターゲットは次のリセレクトまでバスをフリーにします．

(4) イニシエータ・ディテクト・エラー(05H)

ターゲットからのデータ転送中にパリティ・エラーが発生したが，イニシエータがターゲットに再データ転送のリトライ可能であることを知らせるためのメッセージです．

(5) アボート(06H)

イニシエータからターゲットに対して実行中の動作の中止を要求するメッセージです．ターゲットはアボート・メッセージを受け取ると実行中の動作を中断しバス・フリー・フェーズにします．

(6) メッセージ・リジェクト(07H)

イニシエータおよびターゲットから出力されるメッセージで，直前に受け取ったメッセージが不適当であるか実行不可能であることを示します．

(7) ノー・オペレーション(08H)

ターゲットからのメッセージ・アウトの要求に対してイニシエータが意味をもつメッセージをもたない場合にイニシエータが送出するメッセージです．

(8) メッセージ・パリティー・エラー(09H)

イニシエータがターゲットに対して受理したメッセージにパリティー・エラーがあったことを知らせるためのメッセージで，ターゲットはメッセージを再送します．

(9) バス・デバイス・リセット(0CH)

イニシエータがターゲットに対して実行中の命令を中断してターゲット・システムをリセットし，バス・フリー・フェーズに戻ることを要求するメッセージです．

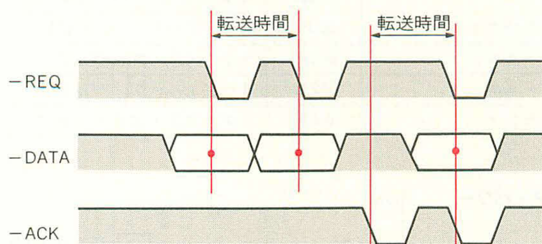
〈図 32〉同期データ転送リクエスト

ビット / バイト	7	6	5	4	3	2	1	0
0	拡張メッセージ(01H)							
1	拡張メッセージ長(03H)							
2	同期データ転送リクエスト・コード(01H)							
3	転送時間(m×4ns) m							
4	REQ/ACK オフセット X							

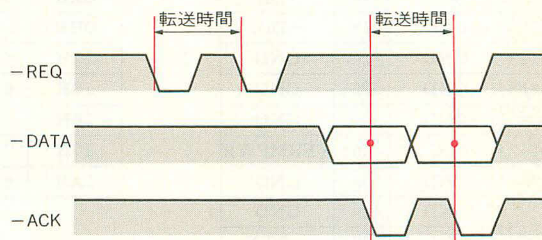
〈表 5〉同期データ転送リクエスト(ターゲット応答)

ターゲット応答	結果
要求以下の REQ/ACK オフセット 要求時間以上の最小転送時間	ターゲット値と同じ REQ/ACK オフセット ターゲット値と同じ最小転送時間
REQ/ACK オフセットが 0 に等しい	非同期転送
MESSAGE REJECT	非同期転送

〈図 33〉同期データ転送の詳細(ターゲット→イニシエータ)



〈図 34〉同期データ転送の詳細(イニシエータ→ターゲット)



内部状態が初期化される点がアボート・メッセージと異なります.

(10) アイデンティファイ(80H, C0H)

イニシエータおよびターゲットから出力されるメッセージで,接続を確立するためのメッセージです.

ビット7はアイデンティファイ・メッセージであることを示し,ビット6はディスコネクト/リセレクション機能の有無を示すビットです.機能がある場合には"1",ない場合には"0"となります.

ビット5からビット3は常に"0"です.

ビット2からビット0はターゲットの論理ユニット番号(LUN)です.

(11) 拡張メッセージ

拡張メッセージとしていくつかのメッセージが規定されていますが,ディスクで一般的に用いられる同期データ転送リクエストを図32に示します.

同期データ転送機能をもつSCSI装置はイニシエータとターゲットの間で同期データ転送リクエストのメッセージを交換することで,転送時間とREQ/ACKオフセットを設定します.

連続した−REQもしくは−ACKの立ち下がりの最小間隔を転送時間といいます.ACK信号をターゲットが受信する前に送出することができるREQ信号の最大数をREQ/ACKオフセットといいます.非同期モードとはREQ/ACKオフセットが0の場合です.オフセットがFFHの場合にはオフセット無制限を示します.

同期データ転送機能をもつイニシエータは最初にターゲットをセレクトしたときに−ATNを表明し,イニシエータが可能なREQ/ACKオフセットと最小転送時間を示す同期データ転送リクエストを送ります.

これに対してターゲットは表5のいずれかの方法で応答し,結果として表に示したモードで転送することになります.

同期データ転送の詳細(オフセット2の場合)についてターゲットからイニシエータへの転送を図33に,イニシエータからターゲットへの転送を図34に示します.また転送時間についてもこれらの図中に示します.

図33においてターゲットは−REQパルスに同期してデータを2個まで先に送出し,データを受け取ったイニシエータはターゲットに−ACKパルスを送出します.ターゲットは−ACKパルスを確認したとき,引き続きデータを送出しますが,−ACKパルスより先に送出できるデータの数は2個までです.

図34においてターゲットは−REQパルスを2個まで先に送出し,データ転送を求めます.イニシエータはこれに対して−ACKパルスに同期してターゲットにデータを送ります.ターゲットは−ACKパルスを確認したとき−REQパルスを必要な数だけ送出しますが,−REQパルス数は−ACKパルス数と同じか多くても2個までです.

電源投入後,バス・デバイス・リセット・メッセージののち,あるいはハード・リセット条件の後では非同期モードです.同期データ転送リクエスト・メッセージ交換はイニシエータとターゲット両方においてセレクション・フェーズに続いて一回だけでなければなりません.

SCSIバス信号

図35にSCSIバスのシングル・エンド型のピン・コネクションを示します.

基本的にはSASIと同様ですが以下の各点でSASI

〈図35〉 SCSI バス信号

1	GND	2	$-DB_0$
3	GND	4	$-DB_1$
5	GND	6	$-DB_2$
7	GND	8	$-DB_3$
9	GND	10	$-DB_4$
11	GND	12	$-DB_5$
13	GND	14	$-DB_6$
15	GND	16	$-DB_7$
17	GND	18	$-DB_P$
19	GND	20	GND
21	GND	22	GND
23	GND	24	GND
25	N.C.	26	TERMPWR
27	GND	28	GND
29	GND	30	GND
31	GND	32	−ATN
33	GND	34	GND
35	GND	36	−BSY
37	GND	38	−ACK
39	GND	40	−RST
41	GND	42	−MSG
43	GND	44	−SEL
45	GND	46	−C/D
47	GND	48	−REQ
49	GND	50	−I/O

と異なります．

(1) −BSY(ビジー)

リセレクション・フェーズにおいてイニシエータもドライブするようになりました．

またアービトレーション・フェーズにおける使用方法が追加されました．

(2) −SEL(セレクト)

リセレクション・フェーズにおいてターゲットもドライブするようになりました．

また，アービトレーション・フェーズにおける使用方法が追加されました．

(3) −ATN(アテンション)

ターゲットはこのラインがアクティブ("L")であることを検出するとメッセージ・アウト・フェーズに入り，イニシエータからのメッセージを受け付けます．アテンションが("H")になったあとのハンドシェイクの終了からターゲットはそのメッセージ・データにしたがってつぎの動作を行います．

(4) TERMPWR(タームパワー)

終端抵抗電源をSCSIデバイスの外部に取り付ける場合，もしくは終端抵抗電源を実装したSCSIデバイスの電源が切断されることがあるシステムではこのラインにSCSIデバイスから電源を供給する必要があります．また電源供給にともないダイオードなどを使って逆流防止を施す必要があり，システムの電源容量が大きく過電流がこのダイオードに流れる恐れのある場合などにはヒューズを取り付ける必要もあります(図36)．

〈図37〉 SCSI コマンド一覧

コード	コマンド名	コード	コマンド名
00H	テスト・ユニット・レディ	(1BH)	(スタート/ストップ・ユニット)
01H	リゼロ・ユニット	25H	リード・キャパシティ
03H	リクエスト・センス	28H	リード・エクステンデッド
04H	フォーマット・ユニット	2AH	ライト・エクステンデッド
(07H)	(リアサイン・ブロック)	2BH	シーク・エクステンデッド
08H	リード	2EH	ライト&ベリファイ
0AH	ライト	2FH	ベリファイ
0BH	シーク	(37H)	(リード・ディフェクト・データ)
12H	インクワイアリィ	C2H	セット・ディスク・パラメータ
15H	モード・セレクト	E0H	チェック・バッファRAM
16H	リサーブ・ユニット	E1H	ライト・ループ・バック
17H	リリース・ユニット	E2H	リード・ループ・バック
1AH	モード・センス	E6H	リクエスト・エラー・ログ

〈図36〉 ターミネータ・パワー供給回路



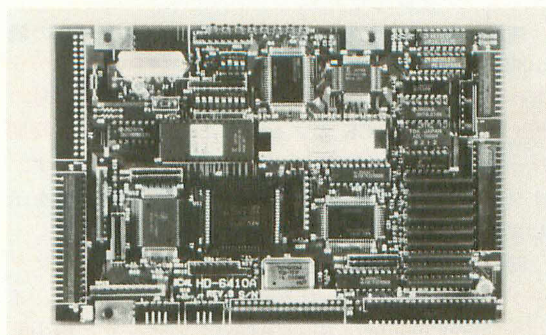
SCSIバスにはシングル・エンド型の他にディファレンシャル型があります．詳しくはコラムを参照してください．

SCSIも1986年に規格化されてから5年が経ち，その間のパーソナル・コンピュータやEWSに代表されるマイクロコンピュータの能力向上はかなりのものです．そこでデータ速度を向上させたSCSI2がSCSIの上位互換として規格化されました．これも基本的にはSCSIと同じですので部分的には順次使用されていくと思われます(7章参照)．

SCSIコマンドの詳細

図37に示した各SCSIコマンドの詳細を示します．

パラメータなど機種依存性のある項目に対しては特に指定しない限りICM製ESDIディスク・コントローラHD-6410A(写真1)を例にとります．



〈写真1〉 HD-6410A

〈図 38〉テスト・ユニット・レディ

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 39〉リゼロ・ユニット

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 40〉リクエスト・センス

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	アロケーション・レングス							
5	0	0	0	0	0	0	0	0

〈図 41〉非拡張センス・データ・フォーマット

バイト＼ビット	7	6	5	4	3	2	1	0
0	AdrVal	センス・コード						
1	LUN			論理ブロック・アドレス(MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス(LSB)							

(1) テスト・ユニット・レディー・コマンド(00H)

ターゲットがレディ状態であるかどうかを調べるための命令です(図 38)．この命令に対してターゲットはリード/ライト動作を含むすべての命令の実行が可能なときにはステータス・コードが正常終了(00H)，またそのほかのときにはチェック・コンディション(02H)となります．

(2) リゼロ・ユニット・コマンド(01H)

この命令を受けたターゲットは装置内部のヘッド位置決め系の再初期化を行い，シリンダ 0 へシークします(図 39)．

(3) リクエスト・センス・コマンド(03H)

SCSI バスの電気的条件

SCSI バスには 2 台から 8 台までの SCSI デバイスがディジィ・チェーンで接続され，両端にターミネータが接続されます．伝送系の電気的条件としては本文中の説明で用いた最大ケーブル長 6m のシングル・エンド型のほかに最大ケーブル長 25m のディファレンシャル型があります．論理的信号のレベルではこれら 2 種類とも共通なのですが，電気的条件がまったく異なるので相互接続は不可能です．

現在シングル・エンド型はドライバ，レシーバ内蔵型の SCSI コントローラを使用することが多く，外付けのドライバ，レシーバが必要なディファレンシャル型は汎用大型機用などきわめて限られたところでしか使用されていません．

● シングル・エンド型

すべての信号は負論理(“L” でアクティブ)です．ドライバとしては 7438，レシーバとしては 74LS14 が電気的条件を満たしています．

▶出力特性

V_{ol}(“L” レベル出力)＝0.0～0.4V

(I_{ol}＝4.8mA のとき V_{ol}＝0.5Vmax)

V_{oh}(“H” レベル出力)＝2.5～5.25V

▶入力特性

V_{il}(“L” レベル入力)＝0.0～0.8V

(V_{il}＝0.4V のとき I_{il}＝0.4mAmax)

V_{ih}(“H” レベル入力)＝2.0～5.25V

入力ヒステリシス＝0.2Vmin

● ディファレンシャル型

＋信号と－信号の 2 本からなり，＋信号の電位が－信号の電位より高いときアクティブ，－信号の電位が＋信号の電位より高いときインアクティブです．

マルチポイント伝送用である EIA 規格の RS485 に適合するドライバ，レシーバを用います．

▶出力特性

I_{ol}(“L”レベル出力)＝55mA のとき

V_{ol}＝2.0Vmax

I_{oh}(“H”レベル出力)＝55mA のとき

V_{oh}＝3.0Vmax

V_{od}(差動電圧)＝1.0Vmin

▶入力特性

I_i＝±2.0mAmax

入力ヒステリシス＝35mVmin

コモン・モード電圧範囲＝－7V～＋12V

ターゲットがチェック・コンディション・ステータスを送出した際にイニシエータが詳細情報を読み取るための命令です(図40).

HD-6410Aでは,図42の22(16H)バイトの拡張センス・データをサポートします.機種によりこれ以上の長さのセンス・データをサポートするものもあります.それらのデータは機種固有のデータであり,一般的には22バイト目以降の追加部分を使用しないほうがいいでしょう.

図40のバイト4のアロケーション・レングスはイニシエータが必要とするセンス・データのバイト数を指定します.HD-6410Aでは00Hから16Hまでが有効で,00Hの場合は4バイトの非拡張センス・データ(SASIコマンドと互換を取るため同じフォーマット)を送出します(図41).非拡張センス・データについてはSASIコマンドの詳細の項を参考にしてください.16H以上の場合には22バイトとします(図42).

非拡張センス・データ(SASI)は論理ブロック・アドレスが21ビットなのに対し,拡張センス・データ(SCSI)は論理ブロック・アドレスを32ビットもっています.

そこで後述する拡張リード・コマンドのように32ビット・アドレスのコマンドを使用する場合には拡張センス・データを用いることになります.

センス・データのバイト0のビット7のAdrValビットは非拡張センス・データのバイト1からバイト3まで,もしくは拡張センス・コードのバイト3からバイト6までの論理ブロック・アドレスが意味をもつ値であるかを示すビットで,AdrVal=“1”のときに論理ブロック・アドレスが意味をもつことを示します.AdrVal=“0”のときは論理ブロック・アドレスが意味をもたないことを示し,論理ブロック・アドレスのすべてのビットは0です.

バイト2のビット3,2,1,0は異常の概要を示すセンス・キーで,表6に示す内容です.

バイト12は異常の詳細を示すセンス・コードで表7に示します.センス・コードは非拡張センス・データの場合のバイト0のビット6からビット0と同じコードです.

チェック・コンディション状態でない場合にリクエスト・センス・コマンドを受け取った場合はセンス・キー=0H,センス・コード=00Hのノー・センスを送出します.

バイト7のアディショナル・センス・レングスはバイト8以降のセンス・データのバイト数を示します.

(4) フォーマット・ユニット・コマンド(04H)

磁気ディスク媒体のフォーマットを要求する命令で

〈図42〉拡張センス・データ・フォーマット

バイト＼ビット	7	6	5	4	3	2	1	0
0	AdrVal	1	1	1	0	0	0	0
1	LUN			0	0	0	0	0
2	0	0	0	0	センス・キー			
3	論理ブロック・アドレス(MSB)							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス							
6	論理ブロック・アドレス(LSB)							
7	アディショナル・センス・レングス							
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
12	センス・コード							
13	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0

〈表6〉センス・キー

センス・キー	内　容
0H	No Sense：装置に異常がなかったことを示す
1H	Recovered Error：装置自体のリトライ機能によりコマンドが正常に終了したことを示す
2H	Not Ready：装置自体に異常が発生し,オペレータ介入が必要であることを示す
3H	Medium Error：リード/ライト動作時において媒体に起因するエラーが発生し,リトライでエラーが回復できなかったことを示す
4H	Hardware Error：装置のハードウェアに障害に発生し,コマンドの実行が正常終了しなかったことを示す
5H	Illegal Request：イニシエータからのコマンド・データに異常があり,コマンドの実行ができなかったことを示す
6H	Unit Attention：装置がリセットされたことを示す
7H	Write Protected：ライト・プロテクト状態にあるときにライト動作を含むコマンドを受け取ったことを示す
BH	Aborted Command：ターゲットがイニシエータからのコマンドをアボートしたことを示す

〈図43〉フォーマット・ユニット・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0
1	LUN			データ	CmpLst	ディフェクト・リスト・フォーマット		
2	0	0	0	0	0	0	0	0
3	インターリーブ(MSB)							
4	インターリーブ(LSB)							
5	0	0	0	0	0	0	0	0

す．HD-6410A はこの命令が送られても無視しますが，パラメータ・リストは読み取ります．

図 43 のバイト 3 およびバイト 4 のインターリーブはフォーマット時のインタリーブ数を示します．

バイト 1 のビット 4 からビット 0 でフォーマット命令実行時に使用する媒体欠陥を表すディフェクト・リストの形式を設定します．リストの種類としては以下に示す 4 種のディフェクト・リストが使用されます．

① P-List(プライマリィ・ディフェクト・リスト)

装置出荷前に工場内における媒体欠陥検査により検出された媒体欠陥リストで装置内の規定領域に書き込まれています．

〈表 7〉センス・コード

センス・コード	センス・キー	ディスクリプション
00H	0H	No Sense：装置に異常がなくコマンドが正常終了した
01H	4H	No Index/Sector Signal：リード/ライト動作に必要なインデックス/セクタ信号を検出できなかった
02H	4H	No Seek Complete：シーク動作が正常に終了しなかった
03H	4H	Write Fault：ライト動作実行時，ライト系回路に異常があり，ライト動作が正常終了しなかった
04H	2H	Not Ready：装置が起動動作中，もしくは装置に異常が発生した
06H	4H	No Track 0 Found：リゼロ動作が正常に終了しなかった
09H	4H	Track Following Error：ライト中に規定値以上のオフ・トラックが発生した
10H	3H	ID CRC Error：目的のブロックの ID 部の読み出し時に CRC エラーが発生し，回復できなかった
10H	4H	ID Read Error：装置のリード回路に障害が発生し，ID 部が読めない
11H	3H	Unrecovered Read Error：リード時データ部にエラーが発生し，リトライによる回復ができなかった
12H	3H	No ID Address Mark Found：ID 部のアドレス・マークが検出できず，ID 部が検出できなかった
13H	3H	No Data Address Mark Found：データ部のアドレス・マークが検出できず，データ部の読み出しができなかった
14H	3H	No Record Found：目的のブロックを検出できなかった
15H	4H	Seek Error：シーク動作の異常が発生した
17H	1H	Recovered Read Data with Retries：リード・リトライ動作により，データ部が読めた
18H	1H	Recovered Read Data with ECC：ECC 機能によりエラー回復した
19H	3H	Defect List Error：欠陥リストの読み出し時エラーが発生し，正常動作ができない
1AH	5H	Parameter Over Run：パラメータ転送時オーバ・ランが発生した
20H	5H	Invalid Command Code：コマンド・ディスクリプタ・ブロックのコマンド・コードが正常でない，もしくはターゲットがサポートしていないコマンドを受け取った
21H	5H	Illegal Block Address：ターゲットの持つブロック数より，大きな値のブロックが指定された
24H	5H	Illegal Field in CDB：コマンド・ディスクリプタ・ブロックの 2 バイト以降に不正なバイトがあった
25H	5H	Invalid LUN：論理ユニットの指定が不適当である
26H	5H	Invalid Field in Parameter List：コマンドに付随するパラメータに不正な内容があった
27H	4H	Write Protected：ライト・プロテクトに設定されている場合にライト動作を含むコマンドを受け取った
29H	6H	Power on, Reset, Bus Device Changed：装置が初期状態にある(SASI との関係で HD-6410A では発生しない)
2AH	6H	Mode Select Parameter Changed：他のイニシエータにより，モード・セレクト・パラメータが変更された
31H	3H	Medium Format Corrupted：媒体欠陥によりフォーマットが正常終了しなかった
32H	3H	No Defect Spare Location Available：装置内のスペア・ロケーションがなくなり該当ブロックの使用が不可能である
40H	4H	RAM Error：診断時，データ・バッファ RAM に異常があった
41H	4H	Data Path Diagnostic Failure：診断時，ECC 回路に異常があった
42H	4H	Power On Diagnostic Failure：電源投入時の自己診断で装置に異常があった．他のセンス・コードで表示される異常についてはそのコードが優先する
43H	4H	Message Reject Error：イニシエータからメッセージ・リジェクトのメッセージを受け取った
45H	4H	Select/Reselect Failure：リセレクションの不成功
47H	4H	SCSI Interface Parity Error：イニシエータからの情報にパリティ・エラーがあった
48H	BH	Initiator Detected Error：イニシエータから Initiator Detected Error Message を受け取った
49H	BH	Inappropriate/Illegal Massage：メッセージにおいて，装置が受け付けられないメッセージを受け取った

〈図 44〉ブロック・フォーマット
(ヘッダおよび1ディフェクト分)

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス(MSB)							
3	ディフェクト・リスト・レングス(LSB)							
0	ディフェクト論理ブロック・アドレス(MSB)							
1	ディフェクト論理ブロック・アドレス							
2	ディフェクト論理ブロック・アドレス							
3	ディフェクト論理ブロック・アドレス(LSB)							

② C-List(ターゲット・サーティフィケーション・リスト)

フォーマット時に装置が検出した媒体欠陥リストです.

③ D-List(データ・ディフェクト・リスト)

フォーマット・コマンドのデータ・アウト・フェーズ時にイニシエータより送出される媒体欠陥リストです.

④ G-List(グローウン・ディフェクト・リスト)

P-List にない後発媒体欠陥リストです.

図43のバイト1のビット4からビット0と各ディフェクト・リストとの関係を以下に説明します.

ビット4(データ)はフォーマット時にイニシエータがD-Listをもつか否かを示すビットで"1"のときデータ・アウト・フェーズでディフェクト・データの受理を行います. ビット3(Cmp Lst)は上記D-Listが装置のすべての媒体欠陥情報を含むか否かを示すビットで,"1"のときすべての媒体欠陥情報を含むことを示し,"0"のときは追加媒体欠陥情報であることを示します.

バイト1のビット4, 3の組み合わせにより下記の3種の方法でフォーマットします.

ビット 4, 3

0 0:P-List と G-List によりフォーマット

1 0:P-List, G-List および D-List によりフォーマット

1 1:P-List と D-List によりフォーマット

D-Listの形式にはブロック・フォーマット(論理ブロック・アドレスで欠陥位置を示すフォーマット), バイト・フロム・インデックス・フォーマット(インデックス信号から欠陥までのバイト数で欠陥位置を示すフォーマット), 物理的セクタ・フォーマット(シリンダ, ヘッドとセクタ番号で欠陥位置を示すフォーマット)の3種類があります. 図43のバイト1のビット2, 1, 0と各形式の対応を下記に示します.

ビット 2, 1, 0

0 × ×:ブロック・フォーマット

〈図 45〉バイト・フロム・インデックス・フォーマット
(ヘッダおよび1ディフェクト分)

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス(MSB)							
3	ディフェクト・リスト・レングス(LSB)							
0	ディフェクトのシリンダ・ナンバ(MSB)							
1	ディフェクトのシリンダ・ナンバ							
2	ディフェクトのシリンダ・ナンバ(LSB)							
3	ディフェクトのヘッド・ナンバ							
4	インデックスからのディフェクト・バイト(MSB)							
5	インデックスからのディフェクト・バイト							
6	インデックスからのディフェクト・バイト							
7	インデックスからのディフェクト・バイト(LSB)							

(注) ディフェクト・ディスクリプタ(s)のバイト4, 5, 6, 7に示すインデックスからのディフェクト・バイトがFFFFFFFFHのときは該当シリンダ・ヘッドのすべてのブロックをリアサインすることを要求している

〈図 46〉物理的セクタ・フォーマット
(ヘッダおよび1ディフェクト分)

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス(MSB)							
3	ディフェクト・リスト・レングス(LSB)							
0	ディフェクトのシリンダ・ナンバ(MSB)							
1	ディフェクトのシリンダ・ナンバ							
2	ディフェクトのシリンダ・ナンバ(LSB)							
3	ディフェクトのヘッド・ナンバ							
4	ディフェクト・セクタ・ナンバ(MSB)							
5	ディフェクト・セクタ・ナンバ							
6	ディフェクト・セクタ・ナンバ							
7	ディフェクト・セクタ・ナンバ(LSB)							

(注) ディフェクト・セクタ・ナンバがFFFFFFFFHの場合は該当シリンダ・ヘッドのすべてのブロックをリアサインすることを要求している

1 0 0:バイト・フロム・インデックス・フォーマット

1 0 1:物理的セクタ・フォーマット

図44, 図45, 図46に各D-Listのフォーマットを示します.

D-Listの先頭4バイトはディフェクト・リスト・ヘッダです. バイト2, 3にディフェクト・ディスクリプタのバイト数が示されます. ディフェクト・リスト・ヘッダ以降に媒体欠陥位置を示すディフェクト・ディスクリプタが1欠陥当たり4バイトもしくは8バイトで示されます.

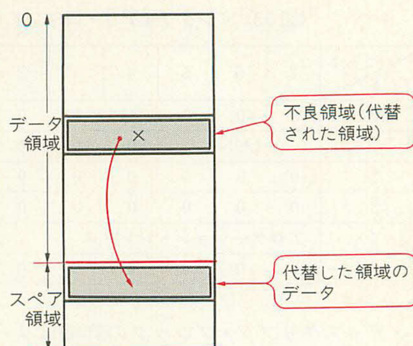
(5) リアサイン・ブロックス・コマンド(07H)

イニシエータがターゲットに対して指定のブロックのリアサイン(異なるロケーションへの再配置)を要求

〈図 47〉 リアサイン・ブロックス・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 48〉
スペア領域



〈図 49〉 リアサイン・ブロック，ディフェクト・リスト
(ヘッダおよび 1 ディフェクト分)

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス(MSB)							
3	ディフェクト・リスト・レングス(LSB)							
0	ディフェクト論理ブロック・アドレス(MSB)							
1	ディフェクト論理ブロック・アドレス							
2	ディフェクト論理ブロック・アドレス							
3	ディフェクト論理ブロック・アドレス(LSB)							

〈図 50〉 リード・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0
1	LUN			論理ブロック・アドレス(MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス(LSB)							
4	転送長							
5	r	c	0	0	0	0	0	0

〈図 51〉 ライト・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0
1	LUN			論理ブロック・アドレス(MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス(LSB)							
4	転送長							
5	r	0	0	0	0	0	0	0

〈図 52〉 シーク・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1
1	LUN			論理ブロック・アドレス(MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス(LSB)							
4	0	0	0	0	0	0	0	0
5	r	0	0	0	0	0	0	0

する命令です．ターゲットはあらかじめ確保されているスペア領域に指定のブロックを再配置します．ここでいうスペア領域とはデータ領域に不良領域がある場合に入れ換えてしまうことで正常な領域のように見せかけるための記憶領域です．この領域は図 48 に示すように一般にディスク・メディア上の内周側に割り当てられています．

本命令の実行後，指定のブロックを含むトラックをスペア・トラックに再配置します．ターゲットにディフェクト・スペアがなくリアサインできない場合にはノー・ディフェクト・スペア・ロケーション・アベイラブルのセンス・データをセットし，チェック・コンディション・ステータスを送出します．

図 49 にリアサイン・ブロック・コマンドでイニシエータから送出されるデータ形式を示します．データ形式はフォーマット・ユニット・コマンドのブロック・フォーマットと同一形式になります．

先頭 4 バイトはディフェクト・リストのバイト数を示すディフェクト・リスト・ヘッダで，バイト 2，3 にディフェクト・ディスクリプタのバイト数を指定します．第 5 バイト以降はディフェクトの位置を示すディフェクト・ディスクリプタで，4 バイトで 1 個のディフェクトの論理ブロック・アドレスを示します．

(6) リード・コマンド(08H)

イニシエータがターゲットに対して磁気ディスク媒体に記録されているデータの転送を要求する命令です．

コマンド・ディスクリプタ・ブロックの論理ブロック・アドレスはデータ転送を要求する先頭のブロックを示し，転送長は転送を要求するブロック数を示します(転送長＝00H は 256 個のブロックの転送要求を示す)．

なお図 50 中の r はリトライ禁止，c は ECC コレクション禁止のビットです．これらは SASI でよく使用されていたモード・ビットで，SCSI 規定ではベンダ・ユニーク・ビットとされています．HD-6410A では SASI との互換性のためこれらのビットをサポートしています．

(7) ライト・コマンド(0AH)

イニシエータがターゲットに対して磁気ディスク媒体へのデータの書き込みを要求する命令です．コマン

〈図53〉インクワイアリ・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	アロケーション・レングス							
5	0	0	0	0	0	0	0	0

ド・ディスクリプタ・ブロックの詳細はリード・コマンドと同じです．

(8) シーク・コマンド(0BH)

イニシエータがターゲットに対してコマンド・ディスクリプタ・ブロックの論理ブロック・アドレスで指定されるブロックの存在するシリンダにシークすることを要求する命令です(図52)．

(9) インクワイアリ・コマンド(12H)

イニシエータがターゲットの属性を知るための命令です(図53)．この命令に対しターゲットはインクワイアリ・データとして図54に示す36(24H)バイトを標準的に出力します．

図53のバイト4のアロケーション・レングスとしては00Hから24Hまでの値が有効で，00Hの場合にはインクワイアリ・データは転送されません．25H以上の値が指定された場合には36バイトで転送を打ち切ります．

図54に示したインクワイアリ・データのバイト0はデバイス・タイプを示します．磁気ディスクはSCSI規定のダイレクト・アクセス・デバイスに該当し，その場合の値は0です．バイト2，3はターゲットがANSIとCCSを満足していることを示します．バイト4のアディショナル・レングスはバイト5以降のバイト数を示し，インクワイアリ・データが36バイトのときには31(1FH)となります．バイト8からバイト15にはベンダ・アイデンティフィケーション“ICM”，バイト16からバイト22にはプロダクト・アイデンティフィケーション“HD-6410 A”(機種により固有の名前)をASCIIコードで示します．バイト32からバイト35はリビジョン・レベルを示す領域で，ターゲットのファームウェア・リビジョンをASCIIコードで示します．

図54のバイト32からバイト35はファームウェア・リビジョン1.10の場合の例を示します．

(10) モード・セレクト・コマンド(15H)

イニシエータがターゲットに対してモードの指定などを行うための命令です(図55)．バイト1のビット4のPF(ページ・フォーマット)はモード・セレクト・ヘッダおよびブロック・ディスクリプタに続いてイニシエータから転送されるパラメータの形式を示すもの

〈図54〉インクワイアリ・データ

バイト＼ビット	7	6	5	4	3	2	1	0
0	ダイレクト・アクセス・デバイス・タイプ=00H							
1	0	0	0	0	0	0	0	0
2	ANSIスタンダード =01H							
3	CCSインプリメンテッド =01H							
4	アロケーション・レングス							
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	“I” =49H							
9	“C” =43H							
10	“M” =4DH							
11	“ ” =20H							
12	“ ” =20H							
13	“ ” =20H							
14	“ ” =20H							
15	“ ” =20H							
16	“H” =48H							
17	“D” =44H							
18	“–” =2DH							
19	“6” =36H							
20	“4” =34H							
21	“1” =31H							
22	“0” =30H							
23	“ ” =20H							
24	“A” =41H							
25	“ ” =20H							
26	“ ” =20H							
27	“ ” =20H							
28	“ ” =20H							
29	“ ” =20H							
30	“ ” =20H							
31	“ ” =20H							
32	ファームウェア・リビジョン“0” =30H							
33	ファームウェア・リビジョン“1” =31H							
34	ファームウェア・リビジョン“1” =31H							
35	ファームウェア・リビジョン“0” =30H							

で，“1”の場合，CCS Rev. 4Bにしたがっていることを示します．

HD-6410Aの場合には“1”，“0”とも以下に示すページ・フォーマットとして命令を実行します．

バイト1のビット0のSP(セーブ・パラメータ)はモード・セレクト・コマンドで転送されるパラメータの取り扱いを指定するビットです．バイト4のパラメータ・リスト・レングスはパラメータのバイト数を示します．

モード・セレクト・コマンドのパラメータにはモード・セレクト・ヘッダ，ブロック・ディスクリプタおよび数種のページ・パラメータがあります(図56，図57，図58)．HD-6410Aではページ・パラメータとし

〈図55〉モード・セレクト・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1
1	LUN			PF	0	0	0	SP
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	パラメータ・リスト・レングス							
5	0	0	0	0	0	0	0	0

〈図57〉モード・セレクト・ブロック・ディスクリプタ

バイト＼ビット	7	6	5	4	3	2	1	0
0	デンシティ・コード＝00H							
1	ブロック・ナンバ(MSB)＝00H							
2	ブロック・ナンバ　　＝00H							
3	ブロック・ナンバ(LSB)＝00H							
4	0	0	0	0	0	0	0	0
5	ブロック・レングス(MSB)＝00H							
6	ブロック・レングス　　＝01H，02H							
7	ブロック・レングス(LSB)＝00H							

て，

1H：エラー・リカバリ・パラメータ

3H：ダイレクト・アクセス・デバイス・フォーマット・パラメータ

4H：リジッド・ディスク・ドライブ・パラメータ

の各パラメータをサポートします．詳細については後述します．

図56にモード・セレクト・ヘッダを示します．

バイト3のブロック・ディスクリプタ・レングスは，モード・セレクト・ヘッダに引き続いて転送されるモード・セレクト・ブロック・ディスクリプタ(**図57**)のバイト数を示すものです．ここが8の場合，次の8バイトがブロック・ディスクリプタです．0の場合にはブロック・ディスクリプタはありません．

図57に示すブロック・ディスクリプタのバイト0はデンシティを規定するものですが，HD-6410Aの場合は固定です．バイト1からバイト3のブロック・ナンバは装置当たりのブロック数を規定する領域です．ブロック・レングス，セクタ/トラックの値によりターゲットの中で自動的に決定されるので，設定はできません．バイト5からバイト7のブロック・レングスはブロック当たりのバイト数を指定する領域ですがHD-6410Aの場合にはボード上のディップ・スイッチで決定されるため設定できません．

モード・セレクト・ヘッダ，ブロック・ディスクリプタに引き続いてページ・ディスクリプタが転送されます．

図58にエラー・リカバリ・パラメータを示します．バイト0のページ・コードは本パラメータ群がエラー・リカバリ・パラメータであることを示します．バイト1のページ・レングスはバイト2以降の本パラメータ群のバイト数を示すものです．バイト2はリトライ方法の詳細を示すものです．

表8にEEC，PER，DTE，DCRビットの組み合

〈図56〉モード・セレクト・ヘッダ

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	ブロック・ディスクリプタ・レングス＝00Hまたは08H							

〈図58〉エラー・リカバリ・パラメータ

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	ページ・コード＝1H					
1	ページ・レングス＝(06H)							
2	0	0	TB	RC	EEC	PER	DTE	DCR
3	リトライ・カウント							
4	コレクション・スパン							
5	ヘッド・オフセット・カウント							
6	データ・ストローブ・オフセット・カウント							
7	リカバリ・タイム・リミット							

〈表8〉リトライ方法の詳細

EEC	PER	DTE	DCR	動作内容
0	0	0	0	リトライを行った後にECC修正を行い，修正後正常終了する．修正不可のときチェック・コンディション・ステータスで異常終了
0	0	0	1	ECC修正を行わない．他の動作は上と同じ
0	1	0	0	リトライ後，ECC修正し，修正可能な場合，すべてのデータの転送後，チェック・コンディション・ステータス，リカバード・センス・キーをセットして修正
0	1	0	1	ECC修正は行わない．他は上と同じ動作
0	1	1	0	リトライを行った後，ECC修正を実行し，チェック・コンディション・ステータス，リカバード・センス・キーをセットしデータ転送を中止
0	1	1	1	ECC修正は行わない．他は上と同じ動作
1	0	0	0	ECC修正を行い，修正不可能の場合リトライを実行．他は0000と同じ
1	1	0	0	ECC修正を行い，修正不可能の場合リトライを実行．他は0100と同じ
1	1	1	0	ECC修正を行い，修正不可能の場合リトライを実行．他は0110と同じ

〈図59〉リザーブ・ユニット・コマンド

ビット / バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0
1	LUN			3rdpty	サード・パーティ・デバイスID			0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図60〉リリース・ユニット・コマンド

ビット / バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1
1	LUN			3rdpty	サード・パーティ・デバイスID			0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図61〉モード・センス・コマンド

ビット / バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0
1	LUN			0	0	0	0	0
2	PCF		ページ・コード					
3	0	0	0	0	0	0	0	0
4	アロケーション・レングス							
5	0	0	0	0	0	0	0	0

〈図62〉エラー・リカバリ・パラメータの例
(変更可能ビットのレポート)

ビット / バイト	7	6	5	4	3	2	1	0
0	1	ページ・コード＝1H						
1	ページ・レングス＝06H							
2	0	0	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1

わせによる動作内容を示します．

▶ DCR(ディスエイブル・コレクション)：エラー修正にECC機能を使用するか否かを指定するビットです．“1”のときはECC機能を使用することを禁止し，“0”のときECC機能の使用を許可します．指定がない場合にはECC機能を用いての修正を行います．

▶ DTE(ディスエイブル・トランスファ・イン・エラー)：エラー発生時点でのイニシエータへのデータ転送について指定するビットです．

▶ PER(ポスト・エラー)：エラー発生時のエラー発生状況の報告方法について規定するビットです．

▶ EEC(イネーブル・アーリ・コレクション)：リトライとECC機能の使用の順序を指定するビットです．

▶ RC(リード・コンティニュ)：“1”のとき，エラー発生時にリトライ/ECC修正を行うことなくデータの転送を指示するビットです．指定がない場合にはリトライ/ECC修正を行います．

▶ TB(トランスファ・ブロック)：“1”のとき，エラー発生に関係なくデータ転送を指示するもので，“0”のときエラー発生ブロックは転送されません．指定がない場合はデータを転送します．

バイト3のリトライ・カウントはECC機能による修正前に実行するリトライ数を指定するバイトです．ターゲットでは標準を8回とします．なお，FFHが指定された場合には無限回となります．

(11) リザーブ・ユニット・コマンド(16H)

複数のイニシエータがある場合にイニシエータがターゲットを占有するために用いる命令です(図59)．この命令によって特定のイニシエータから占有されている場合，ほかのイニシエータからコマンドを受けてもリザベーション・コンフリクト・ステータスにより命令の実行を拒否します．イニシエータ自体がリザーブするユニット・リザベーションとほかの装置への占有を命令するサード・パーティ・リザベーションがあります(マルチプロセスのイニシエータが同一のターゲットを複数のプロセスから同時にアクセスする可能性がある場合にプロセス間でターゲットをロックするために使用されるイクステンド・リザベーションはサポートしない)．

図59のバイト1のビット4は本命令がサード・パーティ・リザベーションであるか否かを示すビットで，“1”のときサード・パーティ・リザベーションであることを示し，ビット3からビット1の3ビットで占有されるべきイニシエータのID(000～111)を示します．

(12) リリース・ユニット・コマンド(17H)

この命令は上で説明したリザーブ・ユニット・コマンドによるターゲットの占有を解除するための命令で，リザーブを行ったイニシエータからのリリース・ユニット・コマンドのみ有効です(図60)．コマンド・ディスクリプタ・ブロックのバイト1の意味はリザーブ・ユニット・コマンドと同じです．

(13) モード・センス・コマンド(1AH)

イニシエータがターゲットの各種パラメータを知るための命令です(図61)．

バイト2のビット5からビット0のページ・コードは転送するセンス・データの内容を示します．バイト4のアロケーション・レングスはセンス・データを受け取るためにイニシエータが用意している領域のバイト数を示します．バイト2のビット7，6のPCF(ページ・コントロール・フィールド)はターゲットから転送

〈図63〉モード・センス・ヘッダ

バイト＼ビット	7	6	5	4	3	2	1	0
0	センス・データ・レングス							
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	ブロック・ディスクリプタ・レングス＝08H							

〈図64〉モード・センス・ブロック・ディスクリプタ

バイト＼ビット	7	6	5	4	3	2	1	0
0	デンシティ・コード＝00H							
1	ブロック・ナンバ(MSB)							
2	ブロック・ナンバ							
3	ブロック・ナンバ(LSB)							
4	0	0	0	0	0	0	0	0
5	ブロック・レングス(MSB)							
6	ブロック・レングス							
7	ブロック・レングス(LSB)							

〈図65〉エラー・リカバリ・パラメータ

バイト＼ビット	7	6	5	4	3	2	1	0
0	1	ページ・コード＝1H						
1	ページ・レングス＝06H							
2	0	0	TB	RC	EEC	PER	DTE	DCR
3	リトライ・カウント＝08H							
4	コレクション・スパン＝08H							
5	ヘッド・オフセット・カウント＝00H							
6	データ・ストローブ・オフセット・カウント＝00H							
7	リカバリ・タイム・リミット＝FFH							

するパラメータのタイプを示すもので，以下の4種があります．

①ビット7，6＝0，0の場合：現在値のレポート

モード・セレクト・コマンドで指定された現在の値を転送します．

②ビット7，6＝0，1の場合：変更可能ビットのレポート

センス・データの変更可能なビットおよびフィールドを“1”とし，変更不可能なビットおよびフィールドを“0”としたデータを転送します．

エラー・リカバリ・パラメータ・ページにおける変更可能ビットの例を図62に示します．図58に示した内容で変更可能な部分を1で示しています．

③ビット7，6＝1，0の場合：規定値のレポート

センス・データのデフォルト値を転送します．

④ビット6，7＝1，1の場合：保存されている値のレポート

センス・データのフォーマット・ユニット・コマンドが完了している値を転送します．

ページ・コードの値が3FHの場合はページ番号順にターゲットがサポートしているすべてのページのデータが転送されます．ページ・コードの値が3FH以外の場合は指定されたページのデータが転送されます．

次にターゲットがサポートするセンス・データのページについて説明します．ターゲットが磁気ディスク装置の場合，下記の3種類のページをサポートするのが一般的です．

1h：エラー・リカバリ・パラメータ

3h：ダイレクト・アクセス・デバイス・フォーマット・パラメータ

4h：リジッド・ディスク・ドライブ・ジオメトリ・パラメータ

ページ・コードが3FHの場合には，モード・センス・ヘッダ，ブロック・ディスクリプタ，エラー・リカバリ・パラメータ，ダイレクト・アクセス・ドライブ・フォーマット・パラメータ，リジッド・ディスク・ドライブ・ジオメトリ・パラメータの順に転送します．

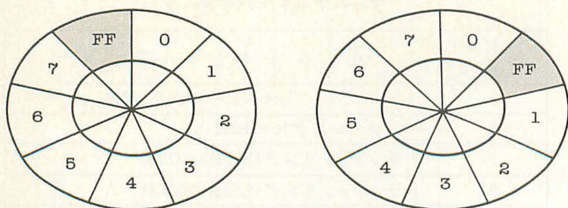
モード・センス・コマンドに対してターゲットはデータ・イン・フェーズで図63に示すモード・センス・ヘッダを転送し，つぎに図64に示すモード・センス・ブロック・ディスクリプタを転送します．

モード・センス・ヘッダのバイト0のセンス・データ・レングスは，モード・センス・コマンドによりデー

〈図66〉ダイレクト・アクセス・デバイス・フォーマット・パラメータ

バイト＼ビット	7	6	5	4	3	2	1	0
0	1	ページ・コード＝3H						
1	ページ・レングス＝16H							
2	トラック/シリンダ(MSB)＝00H							
3	トラック/シリンダ(LSB)＝XXH							
4	代替セクタ/シリンダ(MSB)＝00H							
5	代替セクタ/シリンダ(LSB)＝XXH							
6	代替セクタ/シリンダ(MSB)＝00H							
7	代替セクタ/シリンダ(LSB)＝00H							
8	代替セクタ/ユニット(MSB)＝00H							
9	代替セクタ/ユニット(LSB)＝XXH							
10	セクタ/トラック(MSB)＝00H							
11	セクタ/トラック(LSB)＝XXH							
12	データ・バイト/物理セクタ(MSB)＝00H							
13	データ・バイト/物理セクタ(LSB)＝00H							
14	インターリーブ(MSB)＝00H							
15	インターリーブ(LSB)＝01H							
16	トラック・スキュー・ファクタ(MSB)＝00H							
17	トラック・スキュー・ファクタ(LSB)＝00H							
18	シリンダ・スキュー・ファクタ(MSB)＝00H							
19	シリンダ・スキュー・ファクタ(LSB)＝00H							
20	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0

〈図67〉代替セクタ



(a) 不良スポットがない場合　(b) 不良スポットがある場合

〈図68〉リジッド・ディスク・ドライブ・ジオメトリ・パラメータ

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	ページ・コード＝4H						
1	ページ・レングス＝12H							
2	シリンダ・ナンバ(MSB)＝00H							
3	シリンダ・ナンバ　＝XXH							
4	シリンダ・ナンバ(LSB)＝XXH							
5	ヘッド・ナンバ　＝XXH							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0

〈図69〉スタート/ストップ・ユニット・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1
1	LUN			0	0	0	0	IMMED
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	ST/SP
5	0	0	0	0	0	0	0	0

〈図70〉リムーブ可/不可コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	禁止
5	0	0	0	0	0	0	0	0

タ・イン・フェーズでイニシエータに転送されるセンス・データのバイト数を示し，(全バイト数−1)の値です．これはページ・コードの指定によって異なります．バイト3はブロック・ディスクリプタのバイト数08Hを示します．

モード・センス・ブロック・ディスクリプタは図64に示すような8バイトで構成されます．バイト0のデンシティ・コードは磁気ディスクではデフォルト値は00Hです．バイト1からバイト3のブロック・ナンバはターゲットで使用可能なブロック総数を示し，バイト5，6，7のブロック・レングスは1ブロック当たりのバイト数を示す領域です．

つぎにそれぞれのページについて説明します．

図65に示すエラー・リカバリ・パラメータはターゲットにおけるエラー・リトライの条件を示します．各バイト，ビットの意味はモード・セレクト・コマンドの項を参照してください．

図66にダイレクト・アクセス・デバイス・フォーマット・パラメータを示します．バイト2からバイト9はハンドリング・オブ・ダイレクト・フィールドの領域です．HD-6410Aでは各トラックに1ないし2個の代替セクタを準備し，ディフェクトに対しては同一トラック内でのアロケーションを行います．

トラックをフォーマットしたときに図67(a)に示すように不良セクタがない場合にはいちばん最後のセクタを仮の不良セクタとして使用しません．不良セクタがある場合，不良セクタのところをFFとして飛ばしてセクタを割り付け，有効セクタがすべて正常にアクセスできるようにします．代替セクタ数よりも不良セクタの数が多い場合には同一トラック内へのアロケーションができません．この場合には代替トラックへのアロケーションを行います．

バイト10，11はトラック当たりのセクタ数を示す領域です．HD-6410Aでは代替セクタを含まない値を示します．機種(メーカ)によっては代替セクタを含む総セクタ数を示す場合があります．

バイト12からバイト19のセクタ・フォーマット・フィールドはフォーマット・ユニット時のパラメータを示し，1セクタ当たりのバイト数により決まる値が転送されます．

図68のリジッド・ディスク・ドライブ・ジオメトリ・パラメータはターゲットのシリンダ数，ヘッド数を示します．ほかのバイトについては00Hを転送します．XXHはディスクによってパラメータが変化することを示します．

(14) スタート/ストップ・ユニット・コマンド(1BH)

イニシエータがターゲットに対して装置(スピンドル・モータ)の起動/停止を要求する命令です(図69)．バイト4のビット0のST/SPビットが，

〈図71〉リード・キャパシティ・コマンド

ビット／バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	1
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス(MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス(LSB)							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0

〈図72〉リード・キャパシティ・データ

ビット／バイト	7	6	5	4	3	2	1	0
0	論理ブロック・アドレス(MSB)							
1	論理ブロック・アドレス							
2	論理ブロック・アドレス							
3	論理ブロック・アドレス(LSB)							
4	ブロック・レングス(MSB)							
5	ブロック・レングス							
6	ブロック・レングス							
7	ブロック・レングス(LSB)							

ST/SP＝“1”：スタート・ユニット

ST/SP＝“0”：ストップ・ユニット

という意味をもちます．

バイト1のビット0はイミディエット(IMMED)・ビットを示し，“1”のとき本命令の起動準備が完了した時点でステータスを送出しコマンドを終了します．“0”の場合は本命令の実行が完了するまで終了しません．

なおターゲットは電源ON時に本命令とは関係なくスピンドル・モータを起動します．

(15) リムーブ可/不可コマンド(1EH)

ターゲットに論理ユニットの媒体の取り外しを可能，不可能とすることを要求します(図70)．

バイト4のビット0が“1”のときは媒体の取り外しを禁止し，“0”のときは媒体の取り外しを許可します．MO(光磁気ディスク)の場合などに有効なコマンドです．

(16) リード・キャパシティ・コマンド(25H)

図72に示すリード・キャパシティ・データのバイト0からバイト3にターゲットの最大論理ブロック・アドレスを，バイト4からバイト7にそのブロック・レングスを出力します(図71)．

(17) 拡張リード・コマンド(28H)

イニシエータがターゲットに対して磁気ディスク媒体に記録されているデータの転送を要求する命令です(図73)．リード・コマンド(08H)とは論理ブロック・

〈図73〉拡張リード・コマンド

ビット／バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス(MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス(LSB)							
6	0	0	0	0	0	0	0	0
7	転送長(MSB)							
8	転送長(LSB)							
9	r	c	0	0	0	0	0	0

〈図74〉拡張ライト・コマンド

ビット／バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	0
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス(MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス(LSB)							
6	0	0	0	0	0	0	0	0
7	転送長(MSB)							
8	転送長(LSB)							
9	r	0	0	0	0	0	0	0

〈図75〉拡張シーク・コマンド

ビット／バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	1
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス(MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス(LSB)							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0

アドレスと転送長の指定範囲が異なります．

ブート・ストラップ・ローダのようなプログラムではSASIコマンドのみのHDでも使用できるように08Hのリード・コマンドを使用することがありますが，SCSIデバイスであることが確定している場合には通常28Hのリード・コマンドを使います．本命令の転送長が0000Hの場合はデータ転送の要求がないことを示します．

(18) 拡張ライト・コマンド(2AH)

イニシエータがターゲットに対して磁気ディスク媒体上へのデータ書き込みを要求する命令です(図74)．

〈図76〉ライト・アンド・ベリファイ・コマンド

ビット / バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	0
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス(MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス(LSB)							
6	0	0	0	0	0	0	0	0
7	転送長(MSB)							
8	転送長(LSB)							
9	0	0	0	0	0	0	0	0

〈図77〉ベリファイ・コマンド

ビット / バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス(MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス(LSB)							
6	0	0	0	0	0	0	0	0
7	ベリフィケーション・レングス(MSB)							
8	ベリフィケーション・レングス(LSB)							
9	0	0	0	0	0	0	0	0

ライト・コマンド(0AH)とは論理ブロック・アドレスと転送長の指定範囲が異なります．転送長の指定が0000Hの場合は書き込みがないことを示します．

(19) 拡張シーク・コマンド(2BH)

イニシエータがターゲットに対してバイト2からバイト5で示される論理ブロック・アドレスで指定されるブロックの存在するシリンダにシークすることを要求する命令です(図75)．シーク・コマンド(0BH)とは論理ブロック・アドレスの指定範囲が異なるだけで，同じ動作をします．

(20) ライト・アンド・ベリファイ・コマンド(2EH)

イニシエータがターゲットに対して，磁気ディスク媒体上へのデータの書き込みと書き込みデータの確認を要求する命令です(図76)．ターゲットではECCのチェック機能により書き込みデータの確認を行います．コマンド・ディスクリプタ・ブロックの論理ブロック・アドレスと転送長の意味は拡張ライト・コマンド(2AH)と同じです．

(21) ベリファイ・コマンド(2FH)

イニシエータがターゲットに対して磁気ディスク媒体上に書き込まれているデータの確認を要求する命令です(図77)．ターゲットはECCのチェック機能によってデータ・ベリファイを実行します．

以上でハードディスクにおいて通常定義されているコマンドを説明してきました．しかし，これらのコマンドがSCSIディスクにおいてすべて共通で使用できるというわけでもありません．

できるだけ不特定多数のSCSIハード・ディスクを共通のソフトウェアで使用することを前提とするならばマッキントッシュのOSが行っているようにリード・キャパシティ(25H)，拡張リード(28H)，拡張ライト(2AH)とリクエスト・センス(03H)の4種類のSCSIコマンドのみを使用することを推奨します．

◈参考・引用*文献◈

(1)*Small Computer Interface(SCSI) ANSI X3. 131－1986, ANSI, 1986.

(2) 最新SCSIマニュアル，第2版，CQ出版社，1989.

(3) 事例にみるSCSIインターフェース技術，トリケップス，1989.

代表的なSCSIコントローラ

松村清明

● WD33C93/WD33C93A（ウエスタン・デジタル）

SCSIコマンドをフルサポートしたインテリジェント・チップとして最初に出荷されたものです．内部のマイクロプロセッサがシーケンスをコントロールするので，一連の動作をひとつのコマンドで実行できます（図B）．

▶特徴

- SCSIをフルサポート：アービトレーション（ARBITRATION），ディスコネクト（DISCONNECT），リコネクト（RECONNECT），パリティ（PARITY），同期式のデータ転送で4Mバイト/sで転送可能．オフセットは1から5までプログラムで選択可能．
- 対ホスト用のデータ・バスは8ビット長であり，アドレスとデータ・バスがマルチプレクサされたシステム，またはアドレスとデータ・バスがマルチプレクサされていないシステムのどちらにも対応できる．
- ホスト・アダプタまたはペリフェラル・アダプタのどちらにも使用できる．
- WD33C93には48mAのドライバを含んでおり，SCSIバスに直結可能．
- プログラムドI/O，DMA転送またはダイレクト・バッファ・アクセスのいずれかを選択可能．
- セレクションおよびリセレクション用のプログラマブル・タイマ内蔵．
- “コンビネイション”コマンドはインタラプト・ハンドリングの負担を大いに軽減する．
- 特別な“トランスレイト・アドレス”コマンドはロジカル・アドレスからフィジカル・アドレスへの変更を可能にする．
- 24ビットの転送カウンタ内蔵．
- ＋5VDC単一電源．
- CMOS

● NCR8380

マッキントッシュで使用されているSCSIチップとして有名．データの転送をDMAで行えるロジック以外はプログラムI/Oでコントロールする必要があります．その分シンプルで，CPUが直接コントロールするならオーバ・ヘッドが少ないともいえます．

▶特徴

SCSIインターフェース

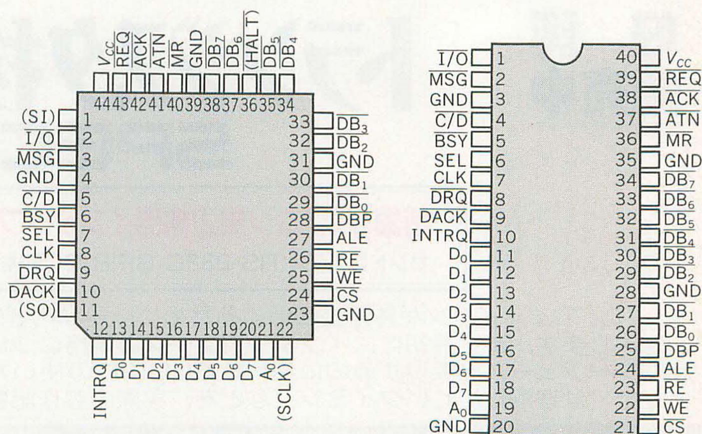
- 非同期1.5MBPS
- イニシエータ，ターゲット・サポート
- パリティ発生/チェック（オプション）
- アービトレーション・サポート
- すべてのバス信号直接制御
- バス・ドライバの内蔵

MPUインターフェース

- メモリ，I/Oマップ・インターフェース
- DMA/プログラムドI/O
- ノーマル，ブロック・モードDMA
- MPU割り込み（オプション）

● MB87034

MB87034は，富士通の一連の（同一コマンド体系の）SCSIコントローラの中の新しいチップです．ヒューレットパッカード社のワークステーションなどで使用されているインテリジェント・コントローラです．－REQ，－ACK端子に3ステート・タイプのドライバ/レシーバを採用し，伝送特性の改善をしてい



(a) 44ピン・フラット　(b) 40ピンDIP

〈図B〉
WD33C93

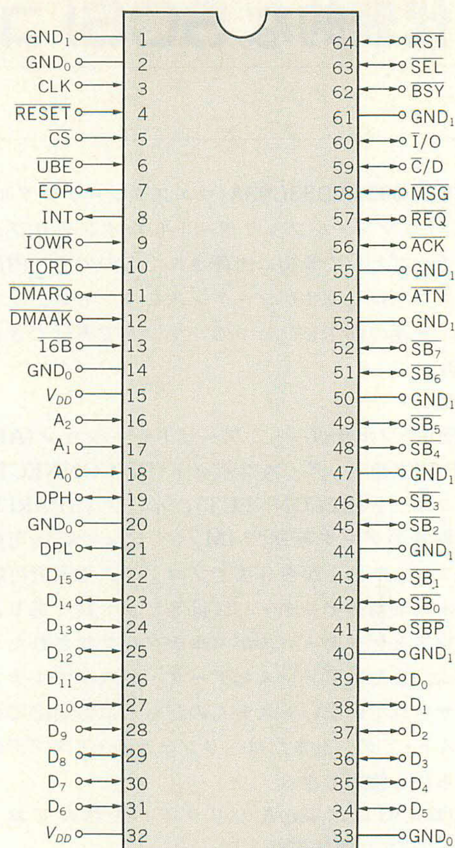
ます．

▶特徴

・ANSI XC3.131-1986 SCSI サポート
・48mA シングル・エンド型　ドライバ/レシーバ内蔵
・−REQ，−ACK 端子にトライステートのドライバ/レシーバ採用
・イニシエータ機能およびターゲット機能の両方をサポート
・同期モード転送による最高 4M バイト/s までのデータ転送が可能
・同期モード転送速度は 4 段階プログラマブル
・同期モード転送オフセット値は 1〜8 バイト・プログラマブル
・8 バイトのデータ・バッファ・レジスタを内蔵
・28 ビット長の転送バイト・カウントにより，一度に 256M バイトのデータ転送が可能
・独立したデータ転送用バス
・2 系統の割り込み信号
・+5V 単一電源，低消費電力動作
・入出力は TTL コンパチブル

● μPD72111

μPD72111 は今回実験に使用した SCSI コントローラです．16 ビット・バスで DMA 可能であり，SCSI バスのフルスピード連続転送を容易に実現できます(図 C)．



〈図 C〉 μPD72111

▶特徴

・ANSI X3.131-1986 に準拠
・2 種類のデータ転送機能
　同期転送(最大 4.0M バイト/s：1 から 8 までのオフセット値を指定可能)
　非同期転送(最大 1.5M バイト/s)
・イニシエータおよびターゲットとして動作可能
・シングル・エンド・タイプの SCSI バス駆動用ドライバおよびシュミット・タイプ・レシーバ内蔵 CPU 側のバス幅を選択可能(16 ビット/8 ビット)
・プログラム転送または DMA 転送を選択可能
・24 ビット転送カウンタ内蔵
・FIFO タイプのデータ・バッファを SCSI バス側および CPU 側の両方に内蔵

第5章

SCSIを使ってハードディスクをつなぐ

ハードディスクを使うための作業はこれだけ

松村清明

それでは実際にSCSIコントローラをPC9801に実装してハードディスクを接続します．次にこのディスクを使用するためのソフトウェアについて説明し，実際にMS-DOSのディスクとして使用する方法を述べます．

実験装置

●SCSIコントローラ

SCSIコントローラ(イニシエータ)・ボードとしてNEC製のET0018(**写真1**)を使用しました．これはμPD72111(NEC製SCSIコントローラ)の評価用ボードです．

ET0018のブロック図を**図1**に，回路図を**図2**に示します．

回路図にはバンク・メモリも記述されていますが，使用しないのでブロック図ではこれをはぶいて説明します．

μPD72111はSCSIバス・ドライバ内蔵のSCSIコントローラですから直接SCSIバスに接続します．

拡張バス・インターフェースから見るとμPD72111とI/Oコントロール用のレジスタ(IC_{19}およびIC_{26})がI/Oデバイスとして見えます．I/Oアドレスはデコード回路(IC_{10}およびIC_{11})により決定されます．

その他，本回路ではDMAとしてDRQ_{30}と$DACK_{30}$を，割り込みとしてIR_{31}を使用します．

●実験装置の接続

SCSIハードディスクとしてICM製SRC-41(**写真2**)を接続しました．PC9801を含めた実験装置の接続を**図3**に示します．

ET0018の設定を**図4**で説明します．

ジャンパ・スイッチJP_1，JP_2はバンク・メモリの使用方法を設定しますが，今回は使用しません．

ジャンパ・スイッチJP_3はμPD72111のINT端子出力をハードウェア割り込みとして使用するか，ハードウェア割り込みを禁止するかを設定するものです．今回はINT端子出力を割り込みとして使用します．具体的には1-2をショートします．

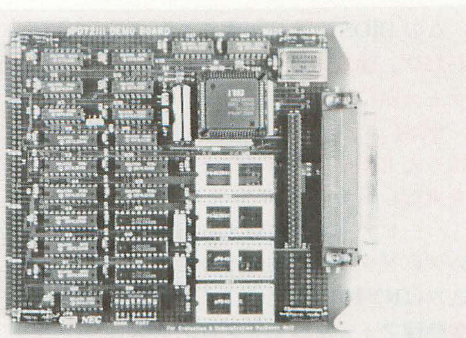
ジャンパ・スイッチJP_4はSCSIバス上のTERMPWRに電源供給するか否かを設定します．今回は電源供給をしません．オープンです．

ジャンパ・スイッチJP_5，JP_6はバンク・メモリを使用するかどうかの設定です．今回は使用しないのでJP_5，JP_6とも2-3ショートです．

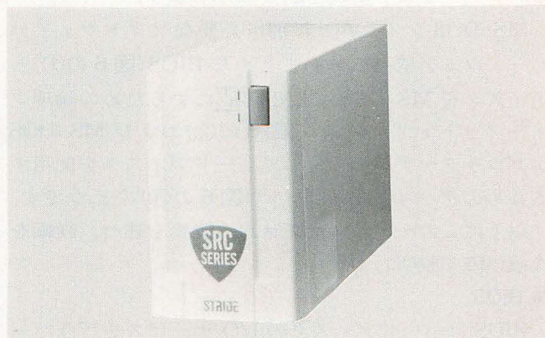
図4のSW_1に関しては**図5**のようにセットします．

ディップ・スイッチSW_1のBANK(1〜3)スイッチではバンク・メモリ・エリアのデコード・アドレスを設定します．バンク・メモリは使用しないので設定の必要はありません．

ディップ・スイッチSW_1のPORT(1〜5)スイッチではμPD72111とI/Oレジスタのデコード・アドレスを設定します．今回はμPD72111のレジスタ・アドレス

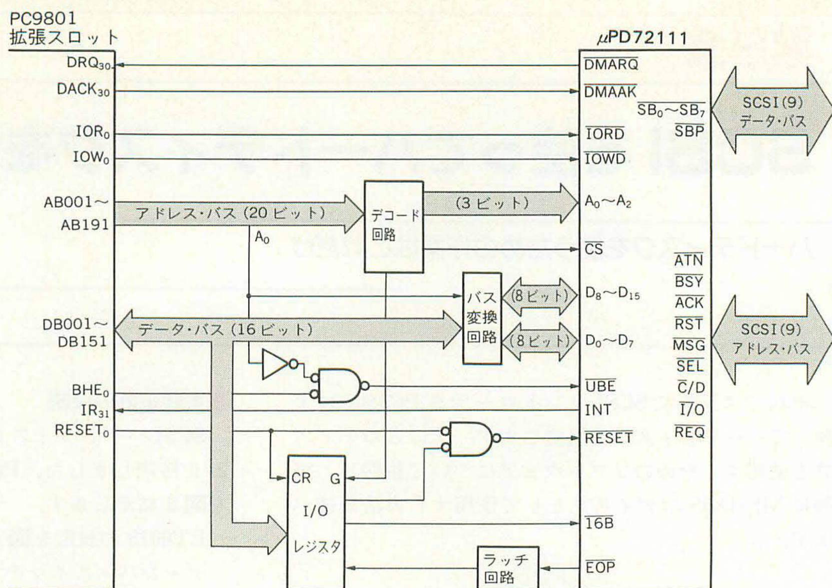


〈写真1〉SCSIコントロール・ボード

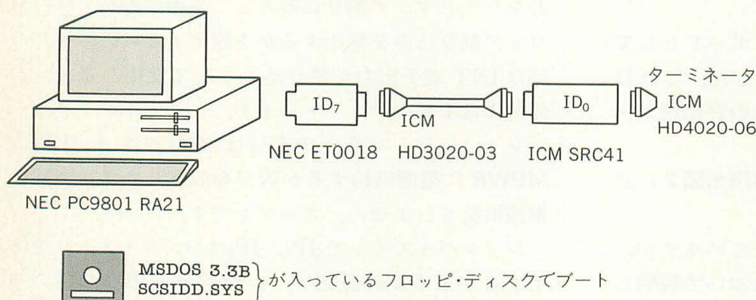


〈写真2〉SRC-41

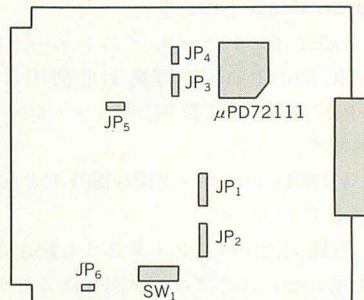
〈図1〉
SCSIコントロール・ボードのブロック図



〈図3〉 実験装置の接続



〈図4〉 ET0018スイッチ配置



は 00D0H～00D7H，I/Oレジスタ・アドレスは 01D0H にセットします．

なお，ET0018の詳しい設定に関しては μPD72111 (SCSIC)の実習テキストを参考にして下さい．

ソフトウェア構成

MS-DOSシステムでハードディスクを使用するためのソフトウェア構成を図6に示します．

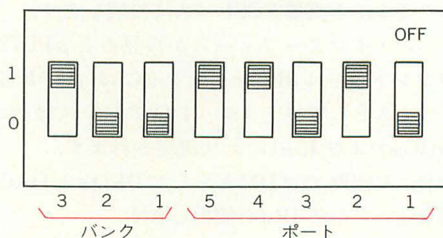
MS-DOSシステムとは別に必要なソフトウェアはハードウェア依存部分をまとめたBIOS(図6の①)とディスクをMS-DOSで使用可能にするための論理フォーマット・プログラム(図6の②)およびMS-DOSがブロック・デバイスとしてハードディスクを使用するためのデバイス・ドライバ(図6の③)の三つです．

以下にこの三つのプログラムの概要を述べ，詳細を次項以降で説明します．

● BIOS

BIOSはハードディスクのI/Oサービスを行うハードウェア依存部分をまとめたプログラムです．

〈図5〉 SW_1の設定



このBIOSをコールすることでハードウェアにとらわれず，論理的なレベルでハードディスクのI/Oアクセスが行えます．

SCSIコマンド・ディスクリプタ・ブロック(CDB)を持ってBIOSコールを行うことによりSCSIコマンドを実行します．

ハードディスクに対するBIOSコールは不特定多数のプログラムから利用できるようにソフトウェア割り込み(INT40H)で行うようにしました．

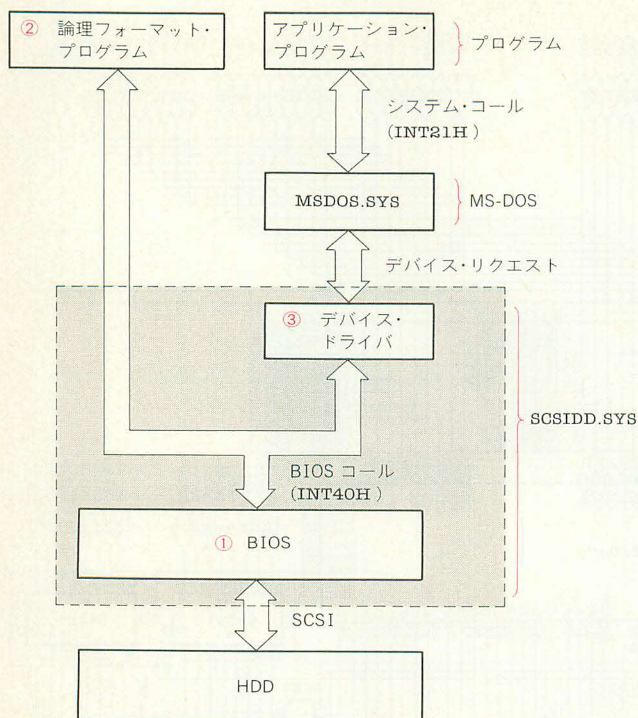
●論理フォーマット・プログラム

論理フォーマット・プログラムはハードディスクに

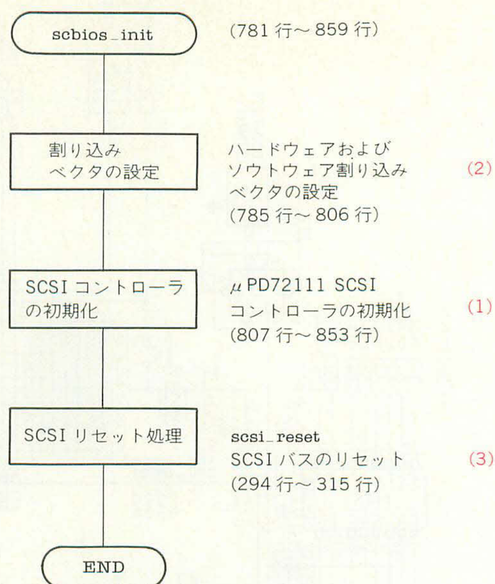
〈図 2〉 SCSIコントロール・ボードの回路図

注：IC_{14}～IC_{17}は今回使用しない

〈図 6〉ソフトウェア構成



〈図 7〉イニシャライズ



MS-DOS のブロック・デバイスとして必要な初期情報を書き込みます．通常，SCSI ハードディスクは物理フォーマットをした状態で出荷されています．

したがって，セクタ単位でアクセスすることは可能ですが，MS-DOS でブロック・デバイスとしてアクセスするにはあらかじめ論理フォーマットしておく必要があります．

●デバイス・ドライバ

デバイス・ドライバは MS-DOS からハードディスクがブロック・デバイスとしてアクセスできるようにするプログラムです．

システム立ち上げ時に組み込まれ，ハードディスクをファイル・デバイスとして使用可能にします．

デバイス・ドライバは BIOS コールを行うので，BIOS もデバイス・ドライバと同時もしくはそれ以前にロードされていることが必要です．今回の例ではデバイス・ドライバと BIOS をひとつのプログラムとして一括でロードしています．

BIOS

SCSI デバイスをアクセスするハードウェア依存部分を BIOS としてまとめています．

この BIOS コールを不特定多数のプログラムから利用できるようにソフトウェア割り込み(INT40H)で行うようにしました．BIOS のプログラム・リストをリスト 1 に示します．また，プログラム中の各処理のフローチャートを図 7～図 13 に示します．

以降の各項で示される行番号はリスト 1 の行番号です．

●イニシャライズ

BIOS を使用する前に scbios_init (781 行～895 行) をコールして各部をイニシャライズすることが必要です．図 7 にフローチャートを示します．

(1) SCSI コントローラのハードウェア割り込みと BIOS エントリのソフトウェア割り込みのベクタを設定

ハードウェア割り込みのマスクも解除します．

(2) μPD72111 SCSI コントローラを初期化

モードの設定，セレクション・タイム・アウトと REQ/ACK タイム・アウトの設定，自分の ID の設定を行い，コントローラがアイドル状態に入るのを待ちます．

(3) SCSI リセット処理

リセット・コマンドを実行し，終了を待ちます．

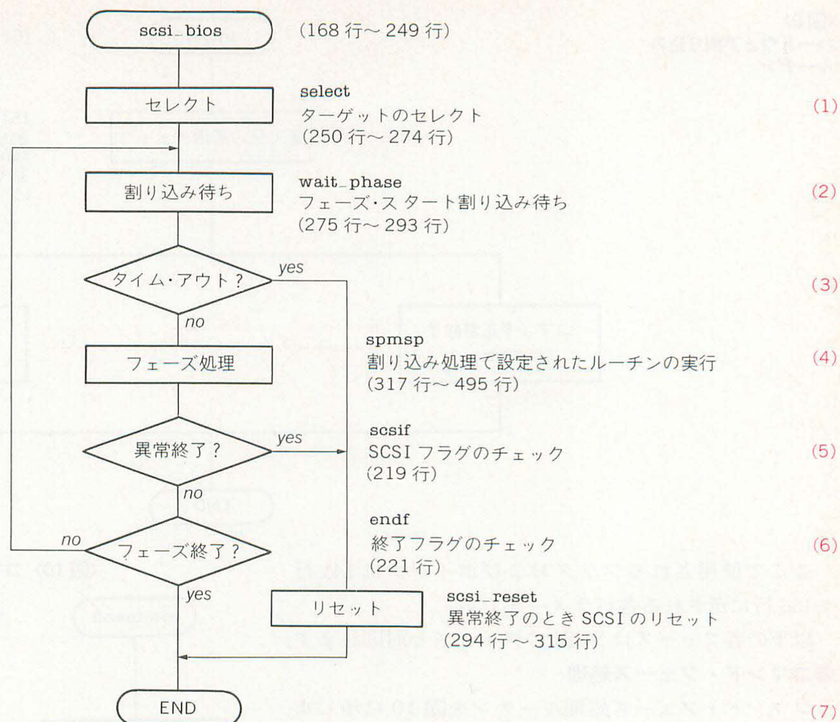
● BIOS エントリ

SCSI BIOS の実行はパラメータをレジスタに設定したのちにソフトウェア割り込みを実行することで行います．結果は AL および AH レジスタにセットされます．

▶入力パラメータ

DS：BX　　CDB(コマンド・ディスクリプ

〈図8〉メイン・ルーチン



	タ・ブロック)のアドレス
DX	CDBのバイト数
ES：DI	データ・バッファ・アドレス
CX	データのバイト数

▶終了パラメータ

AL	SCSIステータス・バイト (133行)
AH	SCSIフラグ (147行～156行)

● BIOSソフトウェア割り込み処理

ソフトウェア割り込み処理ルーチン(メイン・ルーチン)のフローチャートを図8に示します．メイン・ルーチンはハードウェア割り込みによって起動され，各フェーズ処理を行います．

(1) ターゲットをセレクト

SCSIコントローラのアイドル状態を待ちます．アイドル状態になるとBIOSハードウェア割り込みを許可し，ターゲットIDをセット．そしてセレクト・コマンドを発行，タイム・アウト・カウンタをセットし，割り込み発生を待ちます．

(2) フェーズ・スタート割り込みを待つ

(3) タイム・アウトであればSCSIリセット処理を行い，タイム・アウト・フラグを終了パラメータにセットして終了

(4) 各フェーズの処理

フェーズ・スタート割り込みがあるとSCSIフェーズ処理スタート・ポインタに必要な処理ルーチンのポインタがセットされるので，そのフェーズ処理を実行します．SCSIフェーズ処理スタート・ポインタのセットは次の項で述べるBIOSハードウェア割り込みで行います．

フェーズ処理はコマンド・フェーズ処理，データ・フェーズ処理，ステータス・フェーズ処理，メッセージ・イン・フェーズ処理があります．これらについては後述します．

(5) 実行されたフェーズの終了状態をチェックし，異常終了であればSCSIリセット

SCSIリセット処理はSCSIコントローラのアイドル状態を待ち，リセット・コマンドを発行します．

(6) フェーズ終了でなければ，フェーズ・スタート割り込み待ち(2)に戻る

(7) フェーズ終了ならばBIOS処理を終了

終了処理をしてBIOSハードウェア割り込みを禁止し，SCSIステータス，SCSIフラグをAXレジスタに返り値としてセットします．

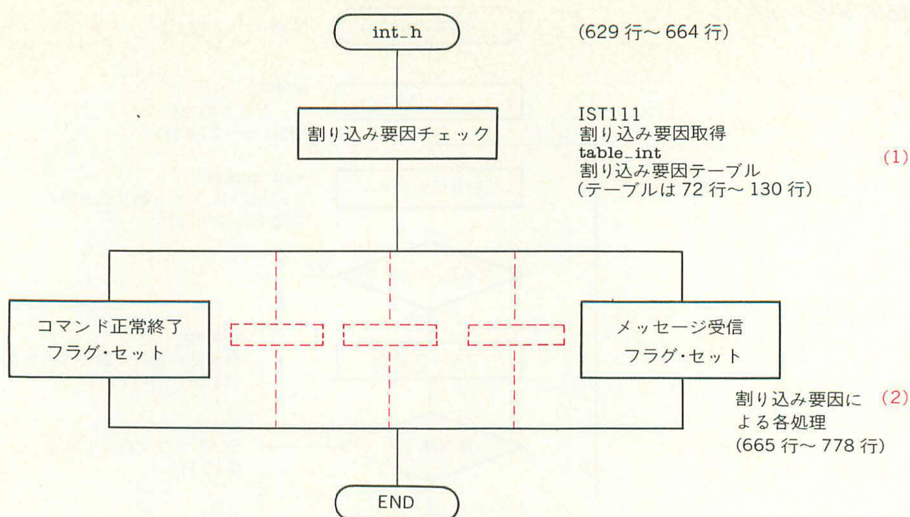
● BIOSハードウェア割り込み処理

ハードウェア割り込み処理ルーチンを図9に示します．このルーチンはμPD72111 SCSIコントローラからの割り込みに対する処理を行います．

(1) SCSIコントローラからの割り込み要因を調べる

(2) 割り込み要因によって各処理ルーチンに入る

各処理ではメイン・ルーチンの処理を制御するフラグの設定，もしくはメイン・ルーチンで処理されるべきルーチン・ポインタの設定を行います．



〈図 9〉
ハードウェア割り込み
ルーチン

ここで使用されるフラグおよびポインタは 141 行～156 行に示される各パラメータです．

以下の各フェーズは SCSI のフェーズと対応します．

●コマンド・フェーズ処理

コマンド・フェーズ処理ルーチンを図 10 に示します．イニシエータからターゲットへ CDB(コマンド・ディスクリプタ・ブロック)データを転送します．

(1) SCSI コントローラのアイドル状態を待つ

(2) CDB のトランスファ・カウントを設定し，トランスファ・コマンドを発行

(3) CDB の長さのデータを出力

REQ/ACK を見ながら CDB データを 1 バイトずつデータ FIFO レジスタに書き込み，ターゲットに転送します．

(4) コマンドの正常終了を待つ

正常終了で戻ってこないときはタイム・アウト・エラーで終了します．

●データ・フェーズ処理

データ・フェーズ処理ルーチンを図 11 に示します．このルーチンはデータ・フェーズ・モードの設定にもとづいてデータ・イン・フェーズ処理もしくはデータ・アウト・フェーズ処理を実行します．データ・イン・フェーズ処理はターゲットからイニシエータへのデータ転送，データ・アウト・フェーズ処理はイニシエータからターゲットへのデータ転送です．

(1) DMA の転送モードを設定

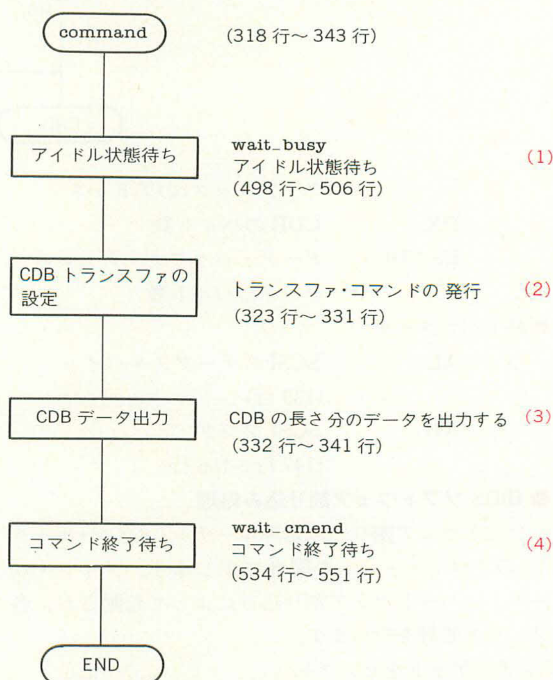
DMA のチャネル 3 に対してシングル・モード，アドレス・インクリメント，オート・イニシャライズ禁止のモードと，ライト転送かリード転送かを設定します．

(2) DMA コントローラを設定

DMA バンク・レジスタのセット，DMA ベース・アドレスのセット，DMA カウンタのセットを行います．

(3) データの長さをトランスファ・カウンタに設定し，

〈図 10〉 コマンド・フェーズ処理



トランスファ・コマンドを発行

(4) コマンドの終了を待つ

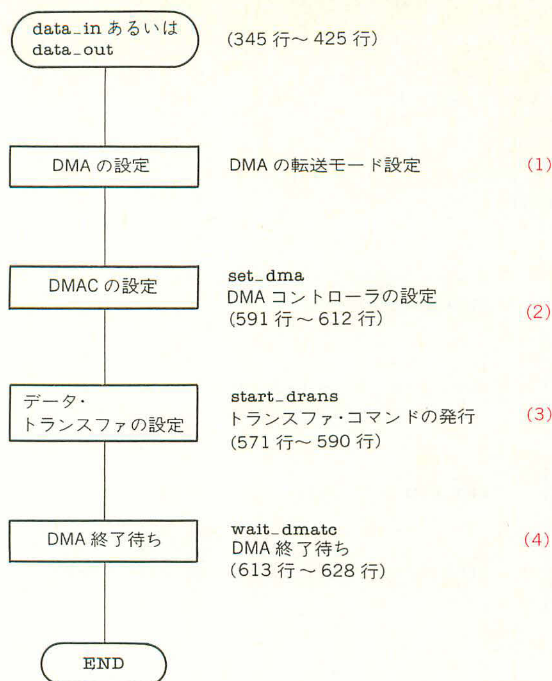
タイム・アウト・カウントをセットし，コマンドの正常終了を待ちます．正常終了で戻ってこない場合はタイム・アウト・エラーで終了します．

● ステータス・フェーズ処理

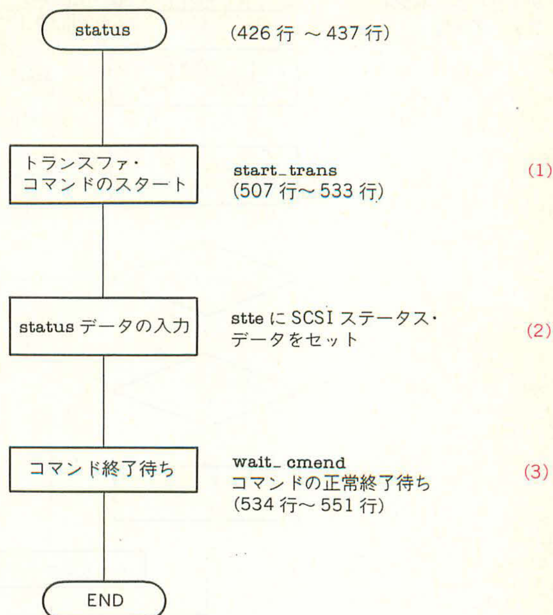
ステータス・フェーズ処理ルーチンを図 12 に示します．ターゲットからステータス・バイトを読みとります．

(1) トランスファ・コマンドをスタート

(2) ステータス・データを入力



〈図 11〉データ・フェーズ処理



〈図 12〉ステータス・フェーズ処理

データ FIFO レジスタから 1 バイトをリードし，リードしたデータをステータス領域にセットします．設定時間内にステータスを受け取らなかった場合はタイム・アウト・エラーになります．

(3) コマンドの正常終了を待つ

タイム・アウト・カウントをセットし，コマンドの正常終了を待ちます．正常終了で戻ってこない場合はタイム・アウト・エラーで終了します．

●メッセージ・イン・フェーズ処理

メッセージ・イン・フェーズ処理ルーチンを**図 13** に示します．ターゲットからのメッセージを読みとります．

(1) トランスファ・コマンドをスタート

(2) ターゲットからメッセージ・データを入力

データを FIFO レジスタから 1 バイト・リードし，リードしたデータをメッセージ・イン領域にセットします．設定時間内にメッセージを受け取らなかった場合はタイム・アウト・エラーになります．

(3) 入力メッセージがコマンド・コンプリート・メッセージであればディスコネクトを待ち，その他のメッセージに対してはメッセージ・リジェクトを出力をするためにＡＴＮを出力

論理フォーマット

論理フォーマットとは MS-DOS のブロック・デバイスとして必要な初期情報をディスクに書き込むことです．デバイス・ドライバを介してハードディスクを使用する前に論理フォーマットを行う必要があります．

論理フォーマット・プログラムの構成は C 言語で記述された部分 FORMATHD.C とアセンブラで記述された部分 BIOSHD.ASM からなります．C 言語部分 FORMATHD.C を**リスト 2** に，アセンブラ部分 BIOSHD.ASM を**リスト 3** に示します．以下で示す行番号は**リスト 2** のものです．

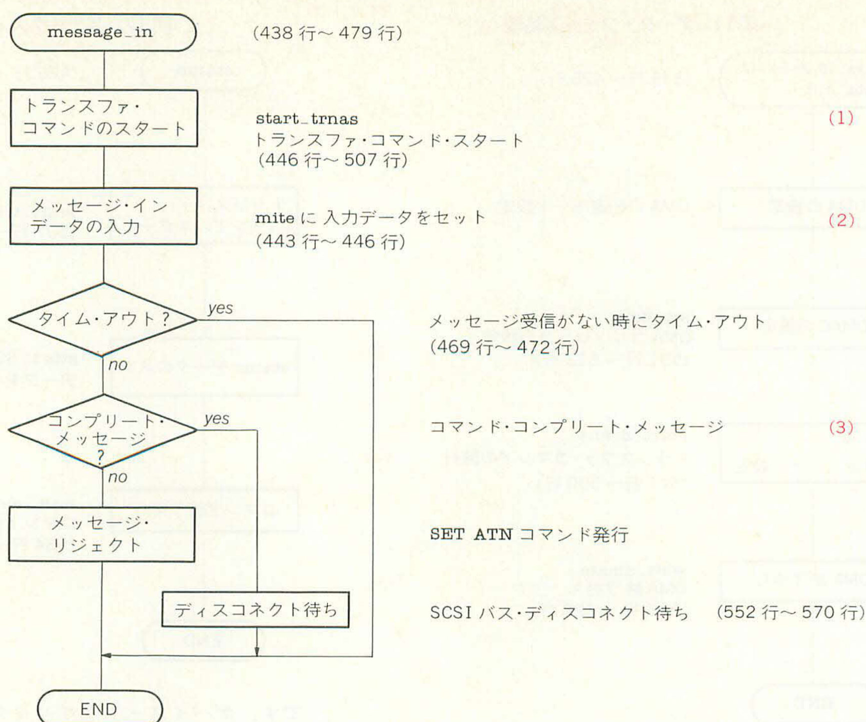
●ブロック・デバイスの初期情報

MS-DOS のブロック・デバイスはデバイス・ドライバを通して見ると固定サイズのブロックの配列になっています．この状態を**図 14** に示します．ブロック配列は先頭から順に予約領域(BIOS パラメータ・ブロック)，FAT 領域，ルート・ディレクトリ領域，ファイル領域の 4 領域に分けて使用されます．

ところで MS-DOS ver.3.X における論理セクタ番号は 16 ビットであつかわれるので論理セクタ番号の最大は 65535 です．したがって 512 バイト/物理セクタとすると，ブロック・デバイスの最大容量は 32M バイトです．そこで 40M バイトのディスクをひとつのブロック・デバイスとしてあつかうためには論理セクタのサイズが最低 1024 バイトである必要があります．ここではふたつの物理セクタ(512 バイト/セクタ)を合わせてひとつの論理セクタとして使用することで 64M バイトまでをひとつのブロック・デバイスとしてあつかえるようにしました．

FAT 領域とルート・ディレクトリ領域は初期化されていることが必要です．また，一般的に論理セクタ 0 (予約領域)に BPB(BIOS パラメータ・ブロック，後

〈図 13〉
メッセージ・イン・
フェーズ処理



〈図 14〉
ブロック・デバイスの
レイアウト

論理セクタ番号	領域名称	初期設定データ
0	予約領域(BPB) 1 セクタ(2 セクタ)	
1 2 3 4	第 1 の FAT 領域 4 セクタ(8 セクタ)	F8 FF FF 00 ～ 00
5 6 7 8	第 2 の FAT 領域 (第 1 の FAT 領域のコピー) 4 セクタ(8 セクタ)	F8 FF FF 00 ～ 00
9 ≀ 104	ルート・ディレクトリ領域 96 セクタ(192 セクタ)	00 00 ～ 00 00
105 ≀ 39599	ファイル領域	

・512 バイト/セクタの物理セクタふたつをひとつの論理セクタとした場合の 40M バイト・ディスクのレイアウト例
・セクタの( )は物理セクタ数

述)を設定します．以下，これらの論理セクタについて述べていきます．

予約領域は論理セクタ 0 から始まります．MS-DOS は必ずしも予約領域を必要とするわけではありません．しかし，BPB(BIOS パラメータ・ブロック)を論理セクタ 0 に入れておくとデバイス・ドライバが最初に論理セクタ 0 を読むことで任意長のディスクに対応できるため論理ブロック 0 を予約領域とし，この中に BPB を入れておくことが一般的です．

● BPB(予約領域内のパラメータ・ブロック)

BPB の構成を表 1 に示します．

BPB は MS-DOS が予約領域，FAT 領域，ルート・ディレクトリ領域，ファイル領域の四つの領域の大きさを決めるためのパラメータです．以下で BPB の各パラメータについて説明します．なお，BPB 内のパラメータで使うセクタとは論理セクタのことであり，ここでは物理セクタを二つ続けたものを論理セクタとしています．

(1) セクタ当たりのバイト数(論理セクタ・サイズ)

MS-DOS がディスク・アクセスするときのセクタ・サイズが何バイトであるかを設定します．

ここでは物理セクタ(512 バイト)×2 ですから，

〈表 1〉
BIOS パラメータ・ブロック
(BPB)の構成

オフセット	パラメータ内容	バイト	設定値
00H	セクタ当たりのバイト数	2	1024
02H	アロケーション・ユニット(1クラスタ)当たりのセクタ数(2のべき乗でなければならない)	1	16
03H	予約セクタ数	2	1
05H	FAT数	1	2
06H	ディレクトリ数	2	3072
08H	論理ボリュームの総セクタ数	2	39600
0AH	メディア・ディスクリプタ・バイト	1	F8H
0CH	FATセクタ数	2	4

1024バイトです.

(2) アロケーション・ユニット当たりのセクタ数

ファイルにディスクの領域を割り当てる単位です.

ここでは16論理セクタです. したがって16Kバイトになります. この単位のことをMS-DOSではクラスタと呼びます.

ファイル長が1バイトのファイルでもディスク上の領域はこの単位で割り当てられています. したがって1バイトから16Kバイトまでのファイルが占有するディスク領域はどれも同じ16Kバイトです.

(3) 予約セクタ数

予約領域としてディスクの先頭から何セクタをMS-DOSのファイル領域外としてあつかうかを設定します.

ここではBPBのために1論理セクタを確保しています. また, ブート可能なデバイスの場合には一般的にブート・ストラップ・ローダをBPBとともにこのセクタに入れておきます.

(4) FAT数

FAT(ファイル・アロケーション・テーブル)のコピーの数です.

FATとはMS-DOSがディスク上のファイルを配置, 管理するためのテーブルです. したがってFATが読み出せないとファイルをまったくアクセスできなくなります.

そこで, 第1のFATが壊れたときのために第1のFATのコピーである第2のFATを用意します. 通常フロッピ・ディスクやハード・ディスクの場合はFATをふたつ用意します. FATの内容はFAT領域の項で説明します.

(5) ディレクトリ数

ルート・ディレクトリの総エントリ数を設定します.

この値により, ルート・ディレクトリの大きさである論理セクタ数が決まります. 計算方法は,

$$\left(\frac{\text{ディレクトリ・エントリ長(32バイト)×ディレクトリ数}}{\text{論理セクタ・サイズ}}\right)$$

=ルート・ディレクトリの論理セクタ数

となります. ただし除算の端数は切り上げます.

(6) 論理ボリュームの総セクタ数

ディスク内のブロック・デバイスとして使用する総論理セクタ数を設定します.

(7) メディア・ディスクリプタ・バイト

フロッピ・ディスクなどでメディアの種類を識別するために使用されます.

ハードディスクの場合特に意味を持ちませんが, 一般的にF8Hを使用するのでここでもF8Hにしました.

(8) FATセクタ数

ひとつのFATが占有するセクタ数を設定します.

FATの配列の要素数は総クラスタ数です. 総クラスタ数は論理ボリュームのセクタ数をクラスタ当たりのセクタ数で割った値になります. この値にFATの要素のサイズ(この場合12ビット)を掛けた値がFATの総ビット数になります. このビット数をセクタまで切り上げた数がFATセクタ数です.

本実験での設定値の場合における計算式を示します.

39600 ÷ 16 = 2475
(論理ボリューム総セクタ数) (クラスタ当たりのセクタ数) (総クラスタ数)

2475 × 12ビット =29700ビット

29700ビット÷8ビット÷1024=4セクタ
(1セクタのバイト数)

なお, 除算の端数は切り上げます.

● FAT領域

FAT(ファイル・アロケーション・テーブル)領域はMS-DOSがディスク上のファイル配置をクラスタ単位で管理するための配列です. 図15に示すようにクラスタには2から順に番号がつけられます. この番号がFATの配列番号に対応します.

Ver.2.X以前のMS-DOSでは配列番号は12ビットで表されていました. したがって配列番号は12ビットで表せる数(0～4095)でした.

後述するようにMS-DOSでは0, 1およびFF0H～FFFHを特別な番号として使用しているので, 実際に使用できる配列番号の数は4078です. したがってクラスタの最大数は4078です.

ディスクの容量が大きいとクラスタ数を4078以下にするためには, クラスタ当たりの大きさを32Kバイト, 64Kバイトなどとかなり大きくしなければなりません.

そこでVer. 3. X以後のMS-DOSでは12ビットの

他に 16 ビット(0～65535)で配列番号を表わすモードが追加されました．この場合も 0，1 および FFF0H～FFFFH を特別な番号として使用しているので，実際に使用できる配列番号の数は 65518 です．ただし，FAT のサイズが大きくなるので MS-DOS のオーバ・ヘッドは重くなります．

今回の実験に使用したディスクの容量は 40 M バイトと比較的小容量なので 12 ビット FAT を使います．

以下でファイル・アクセスの場合を例にとり，簡単に FAT の使用方法を説明します．

アクセスするファイルに対応するディレクトリ・エントリ(ファイル名とその他のパラメータにより構成される 32 バイトのデータ，後述)の中にある開始クラスタ番号がそのファイルのデータ開始クラスタを指します．それと同時にこの番号は FAT 中のエントリ(FAT の配列の要素)の番号を指しており，FAT 中のエントリの内容はそのファイルの次のクラスタ番号を保持しています．

エントリの値が FF8H～FFFH の場合，このエントリがファイル中の最後のクラスタです．また，未使用のクラスタに当たるエントリには 000H が設定されます．

この他にクラスタ番号以外の意味を持つものとして MS-DOS の予約番号である FF0H～FF6H までと不良クラスタを示す FF7H があります．

FAT の先頭のふたつのエントリはメディア・ディスクリプタ・コードである FF8H と FFFH が予約データとして設定されており，クラスタ 0，1 は未使用になります．また，これに続く FAT 領域の初期値としてはその 3 バイトを除いて 000H がセットされます．

●ルート・ディレクトリ領域

ルート・ディレクトリ領域は BPB のディレクトリ数で設定された数分のディレクトリ・エントリを持ったディレクトリのテーブルです．ひとつのディレクトリ・エントリは 32 バイトで構成され，ファイル名，拡張子，属性，日付け，開始クラスタ番号，ファイル容量の各パラメータで構成されています．これを図 16 に示します．

●データ領域

ディレクトリ領域の後ろにクラスタ 2 番の先頭論理セクタが配置されます(前述のとおりクラスタ 0 および 1 は FAT 中の予約コードで使用されるため，データ領域はクラスタ 2 番から始まる)．ここから論理ボリュームの総セクタ数で表される最終セクタまでがデータ領域であり，MS-DOS がディレクトリおよび FAT によって管理し，ファイル領域として使用する領域です．

●論理フォーマット・プログラムの動作

論理フォーマット・プログラムはデバイス・ドライバ内の BIOS を使用しているので，デバイス・ドライバが MS-DOS に組み込まれた環境で実行します．

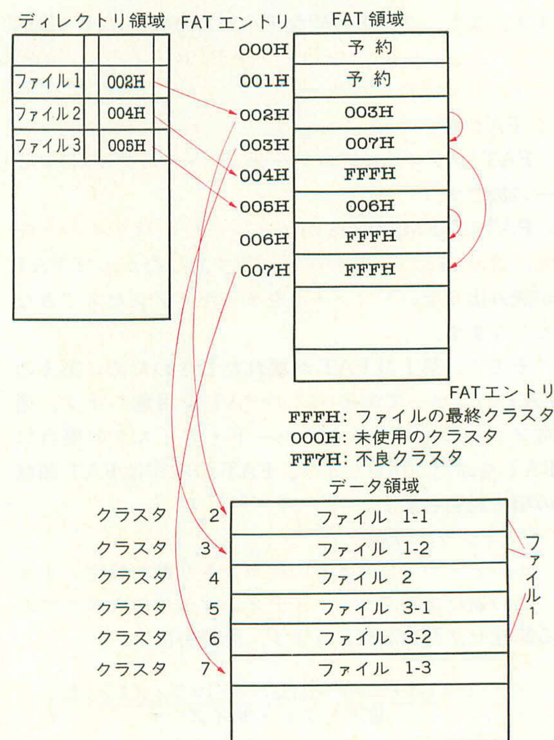
論理フォーマット・プログラムはディスク・ドライブの論理ブロック先頭番地から順に必要なデータを書き込んでいきます．

ICM SRC-41 以外のディスクを使用する場合には BPB(予約領域内のパラメータ・ブロック)の項で説明したように BPB の内容を変更する必要があります．FORMATHD.C(リスト 2)の 6 行～18 行で示されるように，BPB データはフォーマット・プログラム内定数として定義されています．変更する場所は 17 行目の定数です．

64M バイト以下のディスクの場合には論理ボリュームの総セクタ数を容量に応じて変更し，これをもとにクラスタ当たりのセクタ数を変更して FAT セクタ数を計算します．

ディスクの容量が 64M バイト以上の場合には論理

〈図 15〉 FAT とクラスタの関係



〈図 16〉 ディレクトリ・エントリ

00 ～ 07	08～0A	0B	0C～15	16～19	1A 1B	1C ～ 1F
ファイル名	拡張子	属性	予約	日付け	開始クラスタ	ファイル容量

セクタ・サイズを2048バイト以上にすることになります．ただしMS-DOSがブート・アップ時にIO.SYSのパラメータから決定する論理セクタ・サイズ以下でなければなりません．

FATおよびディレクトリの初期化データはBPBの内容によってフォーマット・プログラムが自動的に作るので変更の必要はありません．

論理フォーマット手順のフローチャートを図17に示します．

(1) プログラム中の定数をもとにBPBのデータを作成し，書き込む〔make_bpb()関数〕

BPB書き込み用バッファをクリアし，メーカ名，BPBデータをセットして予約セクタに書き込みます．

(2) BPBの内容をもとにFATデータを作成し，書き込む〔init_fat()関数〕

FAT書き込み用バッファをクリアし，FAT先頭のFAT-ID，予約コード，未使用コードで初期データを作りFAT領域に書き込みます．

(3) BPBの内容をもとにルート・ディレクトリの初期データを作成し，FATの後に書き込む〔init_dir()関数〕

ルート・ディレクトリ書き込み用バッファをクリアし，BPBのディレクトリ数からディレクトリ・セクタ数を求めてルート・ディレクトリ領域に書き込みます．

以上で，ハードディスクをブロック・デバイスとして使用するために必要な初期情報の書き込みが終了します．MS-DOSでブロック・デバイスとして使用するにはこの後リセットしてMS-DOSシステムを再起動する必要があります．

デバイス・ドライバ

MS-DOSは図6に示したようにハードディスクをブロック・デバイスとしてアクセスするためのリクエストをデバイス・ドライバに出します．このリクエストを受け取ったデバイス・ドライバがBIOSを通して実際にハードディスクをアクセスします．

デバイス・ドライバはアセンブラで記述された部分SCSUB.ASMとCで記述された部分SCDV.Cおよび先に述べたBIOS部分SCBIOS.ASMから構成されます．アセンブラ部分SCSUB.ASMをリスト4に，またC言語部分SCDV.Cをリスト5に示します．

●デバイス・ヘッダ

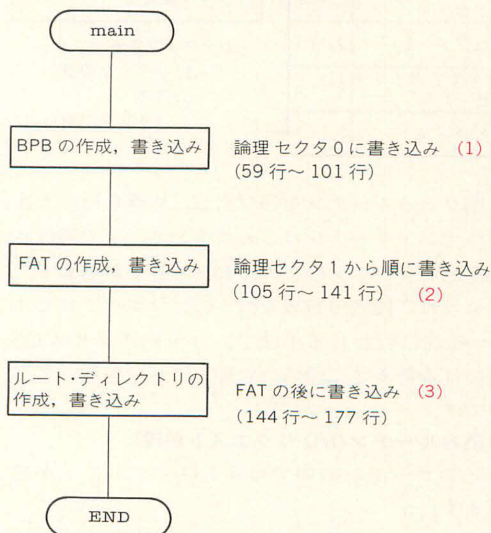
デバイス・ドライバはMS-DOSへの管理情報として先頭にデバイス・ヘッダを持っています．その下にデバイス・ドライバのプログラムがあります．本プログラムのデバイス・ヘッダの値は表2のように設定されています．

次のデバイスへのリンク情報は次のデバイス・ヘッダへのポインタで示し，最後は−1です．本プログラムではデバイス・ヘッダは一つなので−1にします．

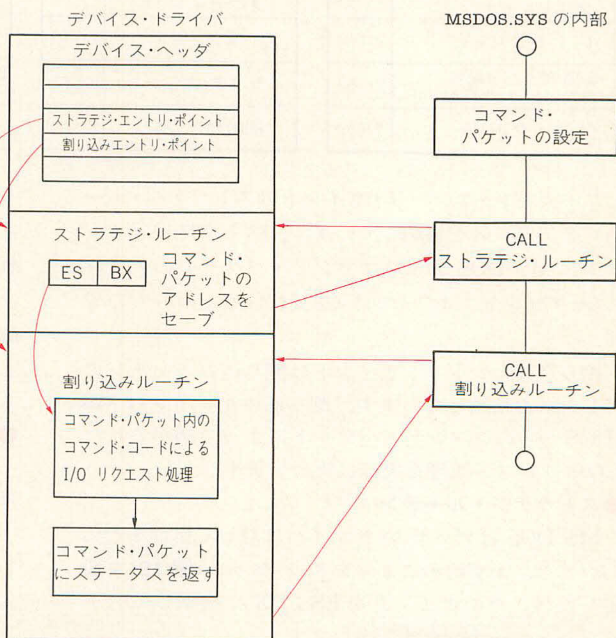
デバイスの属性はブロック型デバイス・ドライバです．ユニット数は1に設定しています．

ストラテジ・エントリ・ポイントと割り込みエントリ・ポイントはプログラムへのポインタです(図18)．実際の動作については後述します．

〈図17〉フォーマット手順



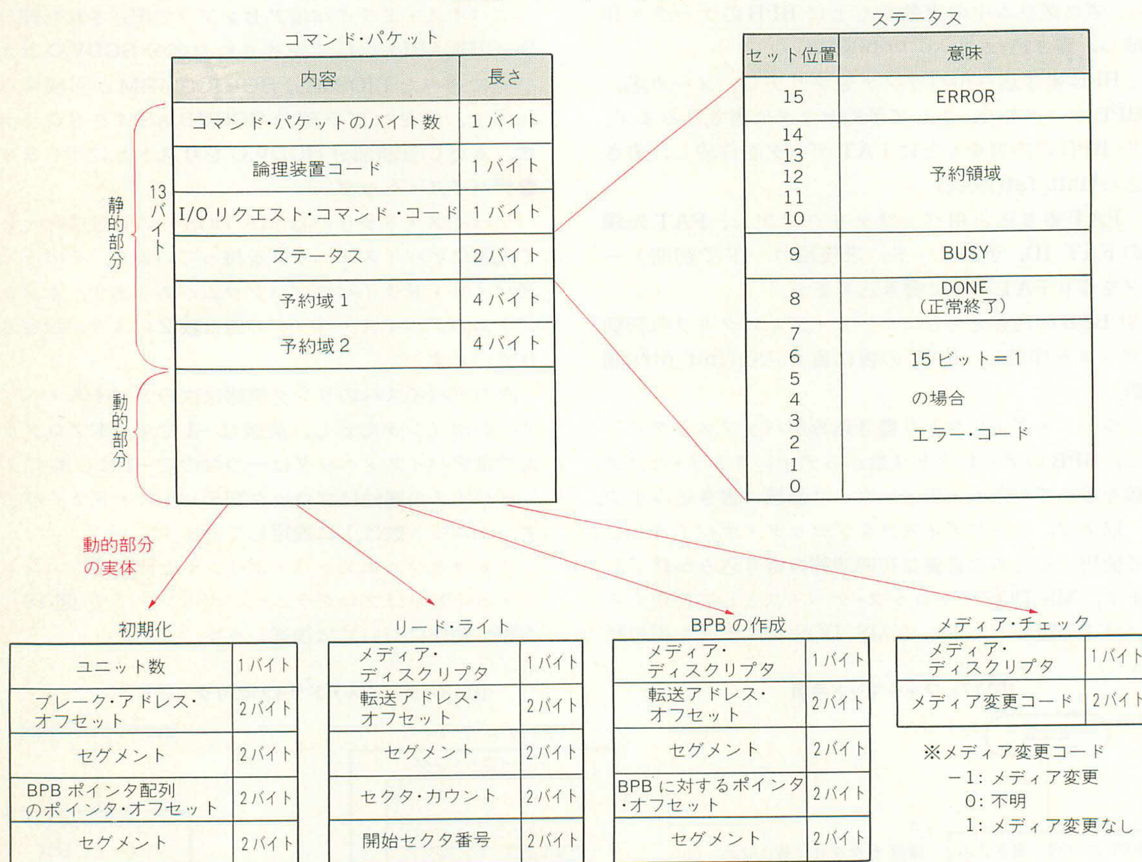
〈図18〉デバイス・ドライバのリクエスト



〈表 2〉デバイス・ヘッダ

	オフセット	内　　容	バイト	設定値
デバイス・ヘッダ	0000H	次のデバイスへのリンク情報	4	−1
	0004H	デバイスの属性	2	2000H
	0006H	ストラテジ・エントリ・ポイント	2	strategy_r
	0008H	割り込みエントリ・ポイント	2	interrupt_r
	000AH	デバイス名またはユニット数	8	1
デバイス本体	0012H 〜 XXXXH	プログラム	可変長	

〈図 19〉コマンド・パケット



ストラテジ・エントリ・ポイントはストラテジ・ルーチンのアドレスを格納しています．ストラテジ・ルーチンは MS-DOS からのコマンド・パケットのアドレス（セグメント：オフセット）を受け取るためのものです．

割り込みエントリ・ポイントは割り込みルーチンのアドレスを格納しています．割り込みルーチンは MS-DOS からのコマンド・パケットによって指示された I/O リクエスト処理を実際に実行します．

●ストラテジ・ルーチン

MS-DOS はデバイス・ドライバに対して処理を行いたいとき，まず始めにコマンド・パケット（図 19）を設定した後，そのポインタを ES：BX に格納してストラテジ・ルーチンを呼び出します．

次に割り込みルーチンを呼び出して処理を行います．

ストラテジ・ルーチンはこのポインタを後で実行される割り込みルーチンのためにセーブしておきます．

このように二段階で処理を行うのはマルチプロセス OS で一般的に行われる手法で，マルチプロセス OS との互換性を考えて二段階の処理ステップを取っているわけです．

●割り込みルーチン（I/O リクエスト処理）

割り込みルーチンの中では各 I/O リクエストが処理されます．

I/O リクエスト処理はコマンド・パケット中のコマンド・コードにより，**表 3** に示す I/O リクエスト処理に分岐して実行されます．

以下の各項で示している行番号は**リスト 5** の行番

〈表3〉I/O リクエスト・コードと処理関数

コード	関数名	処理
00H	init()	初期化
01H	media_check()	メディア・チェック
02H	build_bpb()	BPBの作成
04H	input()	入力
08H	output()	出力
09H	output_v()	ベリファイ付き出力
その他	error_io()	エラー・ステータスの設定

号です．

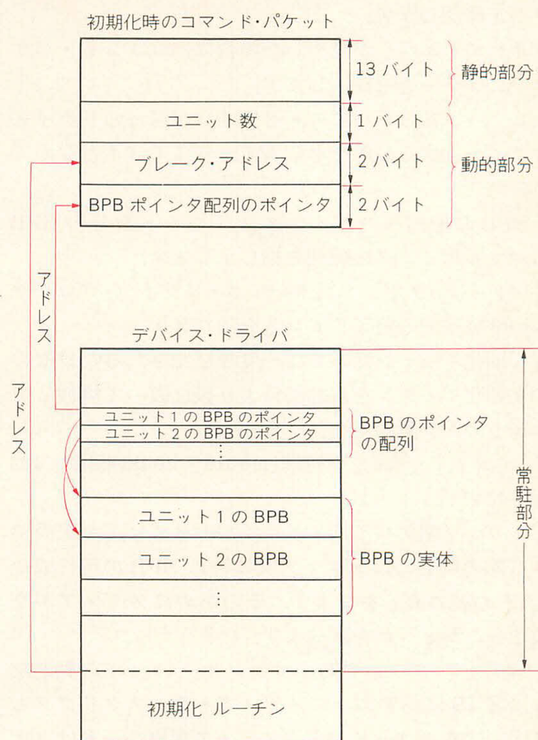
(1) 初期化〔init()関数，360行～419行〕

初期化はI/Oイニシャライズ処理，テスト・レディ処理を実行し，コマンド・パケットにユニット数として1を，加えてブレーク・アドレス(prog_seg, prog_off)とBPB(表1を参照のこと)ポインタ配列のポインタを設定します．この関係を図20に示します．

I/Oイニシャライズ処理はSCSI BIOSの初期化，デバイス・ドライバを組み込んだことの表示を行います．

テスト・レディ処理(611行～688行)はハードディスクがレディ状態であるかどうかSCSI BIOSでテスト・ユニット・レディ・コマンドを送信して確認します．もしレディ状態でなければSCSI BIOSでスター

〈図20〉初期化時のコマンド・パケット



ト・ストップ・コマンドを送信し，もう一度レディかどうかを確認します．

それでもエラーが発生した場合にはコマンド・パケットにエラーをセットします．

ユニット数はデバイス・ドライバが何台のドライブをサポートするのかを示します．ここでは1台なので1です．

BPBの配列へのポインタはユニット数分のBPBに対するポインタの配列を指し示します．

ブレーク・アドレスはデバイス・ドライバ・プログラムの常駐部分の終了アドレスを示します．

初期化ルーチンは始めに一度呼び出されるだけなので初期化ルーチンを常駐部分より後に置いて解放してしまうのが一般的です．

(2) メディア・チェック処理〔media_check()関数，422行〜434行〕

ブロック型デバイス・ドライバでメディアが交換できる物の場合にはメディア交換時にBPBの再作成やFATの読み直しをします．そのためにメディアが交換されたかどうかを調べます．

メディア・チェック時のコマンド・パケットの動的部分は図19に示すようにメディア・ディスクリプタとメディア変更コードです．メディア変更コードはメディアが変更された場合−1，不明の場合0，メディアが変更されない場合には1です．

このドライバ場合，メディアは固定なので常に変更されていない場合の値である1をコマンド・パケットのメディア変更コードに設定し，ステータスにDONEビットをセットします．

(3) BPBの作成〔build_bpb()関数，437行〜454行〕

MS-DOSはメディアにアクセスする前にメディアのBPBの作成を要求します．

BPB作成時のコマンド・パケットの動的部分はメディア・ディスクリプタ，転送アドレス，BPBへのポインタです(図19)．転送アドレスは1セクタ分のバッファのポインタで，作業領域として使うことができます．ただし，本プログラムでは使用していません．

ここでは論理フォーマット・プログラムによってハードディスクの先頭セクタ(セクタ番号0)に書かれたBPBデータを作業領域に読み込み，BPBへのポインタ，BPBへのメディア・ディスクリプタ・バイトをコマンド・パケットにセットします．

(4) 入力処理〔input()関数，471行〜492行〕

論理セクタ単位でデータの読み込みを行います．

コマンド・パケットの動的部分からデータのバッファ先頭アドレス，読み込む論理セクタ数，読み込み開始論理セクタ番号を取り出します．これらをBIOSパラメータに設定し，リード・エクステンド・コマンドをscsi_read()関数で実行してハード・ディスクからデータを読み出します．

コマンドの終了状態をコマンド・パケットのステータスにセットします．正常終了であればDONEビットをセットし，異常終了であればエラー・ステータスをセットします．

(5) 出力処理〔output()関数，495行〜515行〕

論理セクタ単位でデータの書き込みを行います．

コマンド・パケットの動的部分からデータのバッファ先頭アドレス，書き込み論理セクタ数，書き込み開始論理セクタ番号を取り出します．これらをBIOSパラメータに設定し，ライト・エクステンド・コマンドをscsi_write()関数で実行してディスクにデータを書き込みます．

コマンドの終了状態をコマンド・パケットのステータスにセットします．正常終了であればDONEビットをセットし，異常終了であればエラー・ステータスをセットします．

(6) ベリファイ付き出力処理〔output_v()関数，518行〜538行〕

論理セクタ単位でデータのベリファイ書き込みを行います．

scsi_write()関数の代わりにscsi_writev()関数を実行して書き込みと同時にベリファイを行うことを除けば出力処理と同じです．

(7) エラー処理〔error_io()関数，457行〜468行〕

(1)から(6)以外の処理はコマンドが認識できないのでコマンド・パケットのステータスにI/Oリクエスト・コマンドが間違いであるというエラーをセットし，終了します．

MS-DOSから要求されたこれらの処理をBIOSを通してハードディスクに実行してやることによってMS-DOSのファイル・システムとしてハードディスクを使用することが可能になります．

◆引用*・参考文献◆

(1)*NEC，μPD72111 SCSIコントローラ・アプリケーションノート．
(2)*NEC，μPD72111(SCSIC)実習テキスト．
(3) 特集 MS-DOSとデバイスドライバ，トラ技コンピュータ，1989. 11，pp54-91，CQ出版社．
(4) MS-DOSエンサイクロペディア　Volume 1，pp.517-561，アスキー．
(5)*NEC，μPD72111 ユーザーズマニュアル．

〈リスト 1〉 SCBIOS.ASM

```
;  SCSI BIOS FOR uPD72111
;
        page 50,128
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
        public  scbios_init                 ;物理ＢＩＯＳ初期化
_TEXT ENDS
;
_DATA SEGMENT WORD PUBLIC 'DATA'
_DATA ENDS
;
_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS
;
_PEND SEGMENT WORD PUBLIC 'PEND'
_PEND ENDS
;
;       割り込みベクタ
;
vect_seg        segment at 0
                org     0bh*4
INT_VEC_OFF     dw      ?                   ;割り込みベクタ（オフセット値）
INT_VEC_SEG     dw      ?                   ;割り込みベクタ（セグメント値）
;
                org     40h*4
INT_BIOSVEC_OFF dw      ?                   ;ＢＩＯＳ割り込みベクタ（オフセット値）
INT_BIOSVEC_SEG dw      ?                   ;ＢＩＯＳ割り込みベクタ（セグメント値）
vect_seg        ends
;
;       ＳＣＳＩメッセージ
;
CMD_COMP        equu    00h                 ;コマンド・コンプリート・メッセージ
MSG_RJCT        equ     07h                 ;メッセージ・リジェクト・メッセージ
MSG_IDEN        equ     80h                 ;アイデンティファイ・メッセージ
;
;       μＰＤ７２１１１　Ｉ／Ｏ　アドレス
;
TMOD_111        equ     10h                 ;転送モード・レジスタ
BTCL_111        equ     11h                 ;ベース転送カウンタ（下位８ビット）
BTCM_111        equ     12h                 ;ベース転送カウンタ（中位８ビット）
BTCH_111        equ     13h                 ;ベース転送カウンタ（上位８ビット）
SRTOUT_111      equ     21h                 ;セレクション／リセレクション・タイムアウト・レジスタ
RATOUT_111      equ     22h                 ;ＲＥＱ／ＡＣＫハンド・シェーク・タイムアウト・レジスタ
MOD_111         equ     24h                 ;モード・レジスタ
PID_111         equ     25h                 ;物理ＩＤレジスタ
DFL_111         equ     00D0h               ;データＦＩＦＯレジスタ
CST_111         equ     00D2h               ;コントロール・ステータス・レジスタ
ADR_111         equ     00D3h               ;アドレス・レジスタ
WIN1_111        equ     00D4h               ;ウインドウ
DID_111         equ     00D6h               ;デスティネーションＩＤレジスタ
CMD_111         equ     00D7h               ;コマンド・レジスタ
IST_111         equ     00D7h               ;割り込みステータス・レジスタ
IO_PORT         equ     01D0h               ;Ｉ／Ｏレジスタ（ホストアダプタボード）
;
;       その他のチップのＩ／Ｏ　アドレス
;
MODE_37         equ     17h                 ;μＰＤ８２３７　モード・レジスタ
BANK_37         equ     25h                 ;ＤＭＡ　バンク・レジスタ
CLEAR_37        equ     19h                 ;μＰＤ８２３７　クリア・ファースト／ラスト　Ｆ／Ｆ
ADDRESS_37      equ     0Dh                 ;μＰＤ８２３７　ＣＨ３　ベース・アドレス・レジスタ
COUNT_37        equ     0Fh                 ;μＰＤ８２３７　ＣＨ３　ベース・ワード・カウント・レジスタ
MASK_37         equ     15h                 ;μＰＤ８２３７　シングル・マスク・レジスタ・ビット
STATUS_37       equ     11h                 ;μＰＤ８２３７　ステータス・レジスタ
PFCW_59         equ     00h                 ;μＰＤ７１０５９　ＰＦＣＷレジスタ
IMW_59          equ     02h                 ;μＰＤ７１０５９　割り込みマスク・レジスタ
;
DGROUP GROUP _TEXT,_DATA,_BSS,_PEND
        ASSUME  CS:_TEXT,DS:DGROUP,SS:DGROUP,ES:DGROUP
;
;
_DATA SEGMENT WORD PUBLIC 'DATA'
;
;           割り込み要因テーブル
;
table_int   db  00h
            dw  cne                         ;コマンド正常終了処理
            db  08h
            dw  cne                         ;コマンド正常終了処理
            db  20h
            dw  fou                         ;ＦＩＦＯオーバラン／アンダラン処理
            db  28h
            dw  fou                         ;ＦＩＦＯオーバラン／アンダラン処理
            db  22h
            dw  sbpe                        ;ＳＣＳＩバス・パリティ・エラー処理
            db  2Ah
            dw  sbpe                        ;ＳＣＳＩバス・パリティ・エラー処理
            db  23h
            dw  hbpe                        ;ホスト・バス・パリティ・エラー処理
            db  2Bh
            dw  hbpe                        ;ホスト・バス・パリティ・エラー処理
            db  25h
            dw  stoe                        ;セレクト・タイムアウト・エラー処理
            db  26h
            dw  ratoe                       ;ＲＥＱ／ＡＣＫタイムアウト・エラー処理
            db  2Eh
            dw  ratoe                       ;ＲＥＱ／ＡＣＫタイムアウト・エラー処理
            db  30h
            dw  pe                          ;フェーズ・エラー処理
            db  31h
            dw  pe                          ;フェーズ・エラー処理
            db  32h
            dw  pe                          ;フェーズ・エラー処理
            db  33h
            dw  pe                          ;フェーズ・エラー処理
            db  36h
            dw  pe                          ;フェーズ・エラー処理
            db  37h
            dw  pe                          ;フェーズ・エラー処理
            db  80h
            dw  rt                          ;リセット処理
            db  81h
            dw  src                         ;ＳＣＳＩリセット・コンディション処理
            db  90h
            dw  disc                        ;ディスコネクト処理
            db  0A0h
            dw  dops                        ;データ・アウト・フェーズ・スタート処理
            db  0A1h
            dw  dips                        ;データ・イン・フェーズ・スタート処理
            db  0A2h
            dw  cps                         ;コマンド・フェーズ・スタート処理
            db  0A3h
            dw  sps                         ;ステータス・フェーズ・スタート処理
            db  0A6h
            dw  mops                        ;メッセージ・アウト・フェーズ・スタート処理
            db  0A7h
            dw  mips                        ;メッセージ・イン・フェーズ・スタート処理
            db  0C0h
            dw  mess_r                      ;メッセージ受信処理
            db  0C8h
            dw  mess_r                      ;メッセージ受信処理
;
mite        db      ?                       ;メッセージ・イン領域
stte        db      ?                       ;ステータス領域
mote        db      ?                       ;メッセージ・アウト領域
toc         dw      ?                       ;タイムアウト・カウンタ
toc2        dw      ?                       ;タイムアウト・カウンタ（２重ループ用）
dma_bank    db      ?                       ;ＤＭＡバンク値
dma_off     dw      ?                       ;ＤＭＡオフセット値
d_count1    dw      ?                       ;バンク越えするまでの転送数
d_count2    dw      ?                       ;バンク越えしてからの残りの転送数
spmsp       dw      ?                       ;ＳＣＳＩフェーズ処理スタート・ポインタ
cnef        db      ?                       ;コマンド正常終了フラグ
phsf        db      ?                       ;フェーズ・スタート・フラグ
mrf         db      ?                       ;メッセージ受信フラグ
rstf        db      ?                       ;リセット・フラグ
endf        db      ?                       ;終了フラグ
scsif       db      ?                       ;ＳＣＳＩフラグ
                                            ;b0:FIFOｵｰﾊﾞﾗﾝ/ｱﾝﾀﾞﾗﾝ･ｴﾗｰ･ﾌﾗｸﾞ
                                            ;b1:REQ/ACKﾀｲﾑｱｳﾄ･ｴﾗｰ･ﾌﾗｸﾞ
                                            ;b2:ﾌｪｰｽﾞ･ｴﾗｰ･ﾌﾗｸﾞ
                                            ;b3:ﾀｲﾑｱｳﾄ･ﾌﾗｸﾞ
                                            ;b4:SCSIﾊﾞｽ･ﾊﾟﾘﾃｨ･ｴﾗｰ･ﾌﾗｸﾞ
                                            ;b5:メッセージリジェクト
                                            ;b6:SCSIﾘｾｯﾄ･ﾌﾗｸﾞ
                                            ;b7:ﾃﾞｨｽｺﾈｸﾄ･ﾌﾗｸﾞ
;
cdb_add     label   dword                   ;コマンド（ＣＤＢ）アドレス
cdb_off     dw      ?                       ;コマンド（ＣＤＢ）アドレス・オフセット
```

〈リスト 1〉 SCBIOS.ASM(つづき)

```
cdb_seg         dw      ?               ;コマンド(CDB)アドレス・セグメント
cdb_length      db      ?               ;トランスファ・コマンド長
data_length     dw      ?               ;トランスファ・データ長
data_offset     dw      ?               ;データ・バッファ・アドレス・オフセット
data_segment    dw      ?               ;データ・バッファ・アドレス・セグメント
;
_DATA ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
;
;               μPD72111制御処理 物理BIOSエントリ
;
; ターゲットをセレクトした後，μPD72111からの
; フェーズ移行要求により各フェーズ処理に分岐する。
;
;
;  入力：
;       ds:bx   コマンド・アドレス・オフセット & セグメント(CDB)
;       dl      トランスファ・コマンド・カウンタ
;       cx      トランスファ・データ・カウンタ
;       es:di   データ・アドレス・オフセット   & セグメント
;  出力：
;       al      ステータス・コード
;       ah      scsif (リターン・ステータス)
;
scsi_bios               proc    far
        push    bx                      ;AX以外の全レジスタの退避
        push    cx                      ;12[bp]
        push    dx                      ;10[bp]
        push    si
        push    di                      ;6[bp]
        push    ds                      ;4[bp]
        push    es                      ;2[bp]
        push    bp
        mov     bp,sp
        sti
        mov     ax,cs
        mov     ds,ax
        mov     cdb_off,bx              ;CDBアドレス・オフセット
        mov     ax,4[bp]
        mov     cdb_seg,ax              ;CDBアドレス・セグメント
        mov     cdb_length,dl           ;トランスファ・コマンド・カウンタ
        mov     data_length,cx          ;トランスファ・データ・カウンタ
        mov     data_offset,di          ;データバッファアドレス・オフセット
        mov     ax,es
        mov     data_segment,ax         ;データバッファアドレス・セグメント
        mov     cnef,00h                ;全フラグのクリア
        mov     phsf,00h
        mov     mrf,00h
        mov     rstf,00h
        mov     endf,00h
        mov     scsif,00h
;
        call    select                  ;セレクト ターゲット
        jc      scsi_bios_end           ;エラーコードに依る異常終了
scsi_bios_loop:                         ;フェーズ処理ループ
        call    wait_phase              ;フェーズ・スタート割り込み発生待ち
        jc      scsi_bios_rst           ;タイムアウト
        mov     phsf,00h                ;フェーズ・スタート割り込み発生
        call    spmsp                   ;SCSIフェーズ処理
        test    scsif,1fh
        jne     scsi_bios_rst           ;SCSIリセットで異常終了する
        cmp     endf,01h                ;情報転送フェーズ終了か?
        jne     scsi_bios_loop          ;次のフェーズの処理
        mov     endf,00h                ;情報転送フェーズ終了
        jmp     short scsi_bios_enc     ;正常終了
;
scsi_bios_rst:                          ;SCSIをリセットし異常終了する
        call    scsi_reset              ;SCSIリセット
        jc      scsi_bios_end           ;タイムアウト
scsi_bios_enc:
        mov     al,05h                  ;CLEAR FIFO コマンド発行
        mov     dx,CMD_111
        out     dx,al
scsi_bios_end:
        mov     al,80h                  ;デスティネーションIDレジスタに
        mov     dx,DID_111              ;INTM=1 (割り込み禁止),
        out     dx,al                   ;ID=0 (ターゲット) をセット
        mov     al,stte                 ; SCSIステータス
        mov     ah,scsif                ; SCSIフラッグ
        pop     bp                      ; AX以外の全レジスタの復帰
        pop     es
        pop     ds
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        iret
;
scsi_bios       endp
;
;  セレクト ターゲット
;  output
;  cf:on エラーコードに依る異常終了
;
select:
        call    wait_busy               ;アイドル状態を待つ
        mov     al,00h                  ;デスティネーションIDレジスタを選択
        mov     dx,DID_111              ;INTM=0 (割り込み許可)
        out     dx,al                   ;ID=0 (ターゲット) をセット
        mov     al,18h
        mov     dx,CMD_111
        out     dx,al                   ;SELECT コマンド発行
selectw:                                ;セレクト終了待ち
        cmp     scsif,00h
        jne     selecte                 ;エラーコードに依る異常終了
        cmp     cnef,01h
        jne     selectw                 ;セレクト終了待ち
        mov     cnef,00h                ;コマンド正常フラグをリセット
        mov     mote,MSG_IDEN           ;アイデンティファイ・メッセージをセット
        ret
;
selecte:                                ;エラーコードに依る異常終了
        stc
        ret
;
;  フェーズ・スタート・割り込み発生待ち
;  output
;  cf:on タイムアウト この場合タイムアウト・フラグをセット
;
wait phase:
        mov     toc2,0200h              ;タイムアウト・カウンタをセット
        mov     toc,0FFFFh              ;タイムアウト・カウンタをセット
wait_phasew:
        cmp     phsf,01h
        je      wait_phaseg             ;フェーズ・スタート・割り込み発生
        dec     toc
        jne     wait_phasew
        dec     toc2
        jne     wait_phasew
        or      scsif,08h               ;タイムアウト
        stc
wait_phaseg:
        ret
;
;  SCSIリセット
;  output
;  cf:on タイムアウト
;
scsi_reset:
        call    wait_busy
        mov     al,08h
        mov     dx,CMD_111              ;SCSI RESET コマンド発行
        out     dx,al
        call    wait_cmend              ;コマンド正常終了待ち
        jc      scsi_resete             ;タイムアウト
        mov     cx,10
scsi_resetw:                            ;HD内部のリセット終了待ち
        push    cx
        xor     cx,cx
        loop    $
        pop     cx
        loop    scsi_resetw
        clc
scsi_resete:
        ret
;
```

```
subttl  各フェーズの処理
;
;       コマンド・フェーズ処理
;
command:
        call    wait_busy               ;アイドル状態を待つ
        mov     al,BTCL_111             ;ベース転送カウンタ・レジスタを選択
        mov     dx,ADR_111
        out     dx,al
        mov     al,cdb_length           ;トランスファ・コマンド・カウンタ
        mov     dx,WINI_111
        out     dx,al
        mov     al,92h                  ;TRNSFER　コマンド発行
        mov     dx,CMD_111
        out     dx,al
        les     bx,cdb_add              ;コマンド(CDB)アドレス
cmd_drq:                                ;SCSIコマンド出力ループ
        call    wait_drq
        jc      cmd_end                 ;REQ/ACKタイムアウト・エラー
        mov     al,es:[bx]              ;SCSIコマンドデータ
        mov     dx,DFL_111
        out     dx,al                   ;データFIFOレジスタ・ライト
        inc     bx                      ;SCSIコマンドのアドレスをインク
        dec     cdb_length              ;トランスファ・コマンド・カウンタ
        jne     cmd_drq                 ;次のバイトへ
        call    wait_cmend              ;コマンド正常終了待ち
cmd_end:
        ret
;
;       データ・イン・フェーズ処理
;
data_in:
        mov     al,47h
                                        ;シングル・モード選択
                                        ;アドレス・インクリメント選択
                                        ;オート・イニシャライズ禁止
                                        ;ライト転送
                                        ;チャネル3選択
        jmp     short   data_io         ;データ・イン・アウト共通ルーチン
;
;       データ・アウト・フェーズ処理
;
data_out:
        mov     al,4Bh
                                        ;シングル・モード選択
                                        ;アドレス・インクリメント選択
                                        ;オート・イニシャライズ禁止
                                        ;リード転送
                                        ;チャネル3選択
;
;       データ・イン・アウト共通フェーズ処理
;  input
;  al:DMAモード
;
data_io:
        out     MODE_37,al              ;DMAC　モード・レジスタをセット
        in      al,STATUS_37            ;DMAC　ステータス・レジスタをダミー・リード
        mov     ax,data_segment         ;転送先のセグメント値を4ビット・シフト
        mov     cl,04h
        shr     ax,cl
        mov     dma_bank,ah             ;バンク値を退避
        mov     ax,data_segment         ;転送先のセグメント値を4ビット・シフト
        mov     cl,04h
        shl     ax,cl
        mov     bx,data_offset          ;転送先のオフセット値
        add     ax,bx                   ;転送アドレスの64Kオフセット
        jnc     data_ioc
        inc     dma_bank                ;DMAバンク・インクリメント
data_ioc:
        mov     dma_off,ax              ;転送アドレスのオフセット値を退避
        mov     bx,0000h
        sub     bx,ax
        mov     d_count1,bx             ;バンク越えするまでの転送数
        cmp     bx,0
        jz      data_ionot              ;バンク越えしない
        cmp     bx,data_length
        jnb     data_ionot              ;バンク越えしない
;
        mov     ax,data_length          ;バンク越えするときのデータ送受信処理
        sub     ax,bx
        mov     d_count2,ax             ;バンク越えしてからの残りの転送数
        mov     al,dma_bank             ;バンクアドレス
        mov     bx,dma_off              ;64Kページのオフセット
        mov     cx,d_count1             ;転送データ数
        call    set_dmac                ;DMACの設定
        call    start_dtrans            ;データ転送コマンドスタート
        call    wait_dmatc              ;DMA終了待ち
        jc      data_ioe                ;REQ/ACKタイムアウト・エラー
        mov     al,dma_bank             ;ターミナル・カウント発生
        inc     al                      ;次のバンクアドレス
        xor     bx,bx                   ;ベース・アドレスに　0
        mov     cx,d_count2             ;残りの転送数を
        call    set_dmac                ;DMACの設定
data_iotc:
        call    wait_dmatc              ;DMA終了待ち
        jc      data_ioe                ;REQ/ACKタイムアウト・エラー
        call    wait_cmend              ;コマンド正常終了待ち
data_ioe:
        mov     al,07h
        out     MASK_37,al              ;チャネル3マスク・ビットをセット
        ret
;
data_ionot:                             ;バンク越えしないときの送受信処理
        mov     al,dma_bank             ;バンクアドレス
        mov     bx,dma_off              ;64Kページのオフセット
        mov     cx,data_length          ;転送データ数
        call    set_dmac                ;DMACの設定
        call    start_dtrans            ;データ転送コマンドスタート
        jmp     data_iotc
;
;       ステータス・フェーズ処理
;
status:
        call    start_trans             ;TRANSFER　コマンドスタート
        jc      stus_end                ;REQ/ACKタイムアウト・エラー
        mov     dx,DFL_111              ;データFIFOレジスタ・リード
        in      al,dx
        mov     stte,al                 ;ステータス領域にセット
        call    wait_cmend              ;コマンド正常終了待ち
stus_end:
        ret
;
;       メッセージ・イン・フェーズ処理
;
message_in:
        call    start_trans             ;TRANSFER　コマンドスタート
        jc      msg_i_end               ;REQ/ACKタイムアウト・エラー
        mov     dx,DFL_111              ;データFIFOレジスタ・リード
        in      al,dx
        mov     mite,al                 ;メッセージ・イン・領域にリード
        mov     cx,0                    ;タイムアウト・カウンタをセット
msg_i_mrf:
        cmp     mrf,01h                 ;メッセージ受信か?
        loopnz  msg_i_mrf
        jne     msg_i_time_out          ;タイムアウト
        mov     mrf,00h                 ;メッセージ受信フラグをリセット
        cmp     mite,CMD_COMP
        je      msg_i_comp              ;コマンド・コンプリート・メッセージ
        cmp     mite,MSG_RJCT
        je      msg_i_rjct              ;メッセージ・リジェクト・メッセージ
                                        ;その他のメッセージは、リジェクト
        or      scsif,20h               ;メッセージ・リジェクト・フラグをセット
        mov     mote,MSG_RJCT           ;メッセージ・リジェクトコードをセット
msg_i_rjct:                             ;メッセージ・リジェクト
        mov     al,03h
        mov     dx,CMD_111
        out     dx,al                   ;SET　ATN　コマンド発行
        mov     al,04h
        mov     dx,CMD_111
        out     dx,al                   ;RESET　ACK　コマンド発行
msg_i_end:
        ret
;
msg_i_time_out:                         ;タイムアウト
        or      scsif,08h               ;タイムアウト・フラグをセット
        jmp     msg_i_end
;
msg_i_comp:                             ;コマンド・コンプリート・メッセージ
```

〈リスト 1〉 SCBIOS.ASM(つづき)

```
        mov     al,04h
        mov     dx,CMD_111
        out     dx,al                   ;RESET ACK コマンド発行
        call    wait_discon             ;ディスコネクト待ち
        jmp     msg_i_end
;
;       メッセージ・アウト・フェーズ処理
;
message_out:
        call    start_trans             ;TRANSFER コマンドスタート
        jc      msg_o_end               ;REQ/ACKタイムアウト・エラー
        mov     al,mote                 ;メッセージ・アウトデータをリード
        mov     dx,DFL_111              ;データFIFOレジスタ・ライト
        out     dx,al                   ;データ(メッセージ)をセット
        call    wait_cmend              ;コマンド正常終了待ち
        jc      msg_o_end               ;タイムアウト
        test    scsif,20h
        je      msg_o_end               ;正常終了
        call    wait_discon             ;メッセージリジェクト
msg_o_end:
        ret
;
subttl  フェーズ処理のサブルーチン
;
;  アイドル状態を待つ
;
wait_busy:
        mov     dx,CST_111              ;コントローラ・ステータス・レジスタ
        in      al,dx
        test    al,80h
        jne     wait_busy               ;CBSY=1 アイドル状態を待つ
        ret
;
;  TRANSFER コマンドスタート
;  output
;  cf:on REQ/ACKタイムアウト・エラー
;
start_trans:
        call    wait_busy               ;アイドル状態を待つ
        mov     al,0D2h
        mov     dx,CMD_111              ;TRANSFER コマンド発行
        out     dx,al
;
;  DRQ=1 を待つ
;  output
;  cf:on REQ/ACKタイムアウト・エラー
;
wait_drq:
        cmp     scsif,00h
        jne     wait_drqe
        mov     dx,CST_111              ;コントローラ・ステータス・レジスタ
        in      al,dx
        test    al,01h
        je      wait_drq                ;DRQ=1 を待つ
        ret                             ;DRQ=1
;
wait_drqe:                              ;REQ/ACKタイムアウト・エラー
        stc
        ret
;
;  コマンド正常終了待ち
;  output
;  cf:on タイムアウト この場合タイムアウト・フラグをセット
;
wait_cmend:
        mov     cx,0                    ;タイムアウト・カウンタをセット
wait_cmendw:
        cmp     cnef,01h                ;コマンド正常終了か
        loopnz  wait_cmendw             ;コマンド正常終了待ち
        jne     wait_cmendt             ;タイムアウト
        mov     cnef,00h                ;コマンド終了フラグをリセット
        ret
;
wait_cmendt:                            ;タイムアウト
        or      scsif,08h               ;タイムアウト・フラグをセット
        stc
        ret
;
;  ディスコネクト待ち
;
wait_discon:
        mov     cx,0
wait_disconw:
        test    scsif,80h
        loopz   wait_disconw            ;ディスコネクト待ち
        je      wait_discont            ;タイムアウト
        and     scsif,7Fh               ;ディスコネクト・フラグをリセット
        mov     al,05h
        mov     dx,CMD_111
        out     dx,al                   ;CLEAR FIFO コマンド発行
        mov     endf,01h                ;終了フラグをセット
        ret
;
wait_discont:                           ;タイムアウト
        or      scsif,08h               ;タイムアウト・フラグをセット
        ret
;
;  データ転送コマンドスタート
;
start_dtrans:
        call    wait_busy               ;アイドル状態を待つ
        mov     bx,data_length
        mov     al,BTCL_111+80h         ;ベース転送カウンタ・レジスタを選択
        mov     dx,ADR_111
        out     dx,al
        mov     al,bl
        mov     dx,WIN1_111
        out     dx,al                   ;トランスファ・データ・カウンタをセット
        mov     al,bh
        out     dx,al
        mov     al,00h
        out     dx,al
        mov     al,12h
        mov     dx,CMD_111              ;TRANSFER コマンド発行
        out     dx,al
        ret
;
;  DMACの設定
;  input
;  al:バンクアドレス
;  bx:64Kページのオフセット
;  cx:count value
;
set_dmac:
        out     BANK_37,al              ;DMA バンク・レジスタをセット
        out     CLEAR_37,al             ;DMAC クリアFF
        mov     ax,bx
        out     ADDRESS_37,al           ;DMAC ベース・アドレスをセット
        mov     al,ah
        out     ADDRESS_37,al           ;DMAC ベース・アドレスをセット
        mov     ax,cx
        dec     ax
        out     COUNT_37,al             ;DMAC カウント・レジスタをセット
        mov     al,ah
        out     COUNT_37,al             ;DMAC カウント・レジスタをセット
        mov     al,03h
        out     MASK_37,al              ;チャネル3マスク・ビット・クリア
        ret
;
;  DMA終了待ち
;  output
;  cf:on REQ/ACKタイムアウト・エラー
;
wait_dmatc:
        cmp     scsif,00h
        jne     wait_dmatce             ;REQ/ACKタイムアウト・エラー
        in      al,STATUS_37            ;DMAC ステータス・レジスタ・リード
        test    al,08h
        je      wait_dmatc
        ret
;
wait_dmatce:                            ;REQ/ACKタイムアウト・エラー
        stc
        ret
;
subttl  ハードウエア割り込み処理
;
;       ハードウエア割り込み処理
```

〈リスト 1〉 SCBIOS.ASM(つづき)

```
;  μＰＤ７２１１１からの割り込み要因を解析し，各処理へ分岐する。
;
int_h   proc
        push    ax                      ;レジスタ退避
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    ds
        mov     ax,cs
        mov     ds,ax                   ;データセグメント＝コードセグメント
        mov     dx,IST_111
        in      al,dx                   ;割り込み要因を取得
        mov     bx,offset DGROUP:table_int-3
int_table:                              ;割り込み要因を解析
        add     bx,3
        cmp     al,[bx]
        jne     int_table               ;次の割り込み要因をチェック
        call    word ptr [bx+1]         ;割り込み処理を呼び出す
        mov     al,63h
        out     PFCW_59,al              ;ＰＦＣＷレジスタにＦＩコマンド発行
        pop     ds                      ;レジスタ復帰
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        iret
;
int_h   endp
;
;       コマンド正常終了処理
;
cne:
        mov     cnef,01h                ;コマンド正常終了フラッグ
        ret
;
;       ＦＩＦＯオーバラン，アンダラン・エラー処理
;
fou:
        or      scsif,01h               ;オーバラン、アンダランエラーフラッグ
        ret
;
;       セレクト・タイムアウト・エラー処理
;
stoe:
        or      scsif,08h               ;セレクト・タイムアウト・エラーフラッグ
        ret
;
;       ＲＥＱ／ＡＣＫタイムアウト・エラー処理
;
ratoe:
        or      scsif,02h               ;ＲＥＱ／ＡＣＫタイムアウト・エラー
        ret
;
;       フェーズ・エラー処理
;
pe:
        or      scsif,04h
        ret
;
;       ＳＣＳＩリセット・コンディション処理
;
src:
        or      scsif,40h               ;ＳＣＳＩリセット・コンディション処理
        ret
;
;       ディスコネクト処理
;
disc:
        or      scsif,80h               ;ディスコネクトフラッグ
        ret
;
;       メッセージ受信処理
;
mess_r:
        mov     mrf,01h                 ;メッセージ受信フラッグ
        ret
;
;       リセット処理
;
rt:
        mov     rstf,01h                ;リセットフラッグ
        ret
;
;       ＳＣＳＩバス・パリティ・エラー処理
;
sbpe:
        or      scsif,10h               ;ＳＣＳＩバス・パリティ・エラーフラッグ
        ret
;
;       ホスト・バス・パリティ・エラー処理
;
hbpe:
        ret
;
;       データ・アウト・フェーズ・スタート処理
;  ｓｐｍｓｐにデータ・アウト・フェーズ処理のポインタをセット
;
dops:
        mov     spmsp,offset _TEXT:data_out
        mov     phsf,01h
        ret
;
;       データ・イン・フェーズ・スタート処理
;  ｓｐｍｓｐにデータ・イン・フェーズ処理のポインタをセット
;
dips:
        mov     spmsp,offset _TEXT:data_in
        mov     phsf,01h
        ret
;
;       コマンド・フェーズ・スタート処理
;  ｓｐｍｓｐにコマンド・フェーズ処理のポインタをセット
;
cps:
        mov     spmsp,offset _TEXT:command
        mov     phsf,01h
        ret
;
;       ステータス・フェーズ・スタート処理
;  ｓｐｍｓｐにステータス・フェーズ処理のポインタをセット
;
sps:
        mov     spmsp,offset _TEXT:status
        mov     phsf,01h
        ret
;
;       メッセージ・イン・フェーズ・スタート処理
;  ｓｐｍｓｐにメッセージ・イン・フェーズ処理のポインタをセット
;
mips:
        mov     spmsp,offset _TEXT:message_in
        mov     phsf,01h
        ret
;
;       メッセージ・アウト・フェーズ・スタート処理
;
;  ｓｐｍｓｐにメッセージ・アウト・フェーズ処理のポインタをセット
;
mops:
        mov     spmsp,offset _TEXT:message_out
        mov     phsf,01h
        ret
;
subttl  物理ＢＩＯＳ初期化ルーチン
;
;  物理ＢＩＯＳ初期化
;
scbios_init     proc    near
        mov     al,00h                  ;８ビット・モードに設定
        mov     dx,IO_PORT
        out     dx,al
;
        cli
sb_init_cstl:                           ;ＣＰＵへの割り込み要求待ち
```

〈リスト 1〉 SCBIOS.ASM (つづき)

```
        mov     dx,CST_111
        in      al,dx                   ;コントロラ・ステータス・レジスタ
        test    al,40h
        je      sb_init_cst1            ;CPUへの割り込み要求待ち
        mov     dx,IST_111              ;割り込みステータス・レジスタ
        in      al,dx                   ;割り込みクリア
        mov     ax,0
        mov     es,ax
        mov     es:INT_VEC_OFF,offset _TEXT:int_h ;ハード割り込みベクタセット
        mov     es:INT_VEC_SEG,cs       ;int 0bh (2ch,2eh)
        mov     es:INT_BIOSVEC_OFF,offset _TEXT:scsi_bios  ;int 40h (100h,102h)
        mov     es:INT_BIOSVEC_SEG,cs   ;BIOS割り込みベクタセット
        in      al,IMW_59
        and     al,0F7h                 ;INT  Bの割り込みマスク解除
        out     IMW_59,al
        sti
;
        mov     al,MOD_111
        mov     dx,ADR_111
        out     dx,al                   ;モード・レジスタを選択
        mov     al,0A0h                 ;DMAデータ転送モード
                                        ;CPUバスパリティOFF
                                        ;SCSIバスパリティON
                                        ;アービトレーション有り
        mov     dx,WIN1_111
        out     dx,al
;
        mov     al,TMOD_111
        mov     dx,ADR_111
        out     dx,al
        mov     al,00h                  ;非同期転送モード
        mov     dx,WIN1_111
        out     dx,al
;
        mov     al,SRTOUT_111           ;セレクト・タイムアウト
        mov     dx,ADR_111
        out     dx,al
        mov     al,1Eh                  ;を250msにセット
        mov     dx,WIN1_111
        out     dx,al
;
        mov     al,RATOUT_111           ;REQ/ACKタイムアウト
        mov     dx,ADR_111
        out     dx,al
        mov     al,080h                 ;を64.7msにセット
        mov     dx,WIN1_111
        out     dx,al
;
        mov     al,PID_111              ;自分のID番号をセット
        mov     dx,ADR_111
        out     dx,al
        mov     al,87h                  ;  ID=7
        mov     dx,WIN1_111
        out     dx,al
;
        mov     cnef,00h                ;コマンド正常終了フラグをリセット
        mov     scsif,00h               ;SCSIフラグ1をリセット
        call    wait_busy               ;アイドル状態を待つ
        mov     al,00h                  ;デスティネーションIDレジスタに
        mov     dx,DID_111              ;INTM=0 (割り込み許可)
        out     dx,al                   ;ID=0 (ターゲット)をセット
        call    scsi_reset              ;SCSIリセット
        mov     al,80h                  ;デスティネーションIDレジスタ
        mov     dx,DID_111              ;INTM=1 (割り込み禁止)
        out     dx,al                   ;ID=0 (ターゲット)をセット
        ret
scbios_init     endp
;
;
_TEXT ENDS
;
;
;
END
```

〈リスト 2〉 FORMATHD.C

```

/* μPD72111フォーマット (FORMATHD. C) */

#include        <stdio.h>

/*  BPBの構造体  */
struct  bpb_st {
        int  b_sector;          /*  1セクタ当りのバイト数           */
        char sector_c;          /*  1クラスタ当りのセクタ数         */
        int  resv_sector;       /*  予約セクタ数                    */
        char fat;               /*  FATの数                        */
        int  dir_no;            /*  ディレクトリ数                  */
        int  tol_sec;           /*  論理ボリュームの総セクタ数      */
        char med_id;            /*  メディア・ディスクリプタ・バイト */
        int  fat_sec;           /*  FATセクタ数                    */
} bpb_h = {
        1024, 16, 1, 2, 3072, 39600, 0xF8, 4
};

/* Jump Code  &  Maker Name  Data */
char    jmak_dat[11] = {
        0xEB, 0x1F, 0x90,                       /* Jump  Code  */
        'I','C','M',' ','D','I','S','C'         /* Maker Name  */
};

#define P_SECTOR_SIZE   512     /* Define  Physical Sector   Size     */
#define BPB_SIZE        13      /* Define  BPB              Data Size */
#define JMAK_SIZE       11      /* Define  Jump & Maker Data Size     */
#define OK              0       /* Define  Success  Flag              */

long    stno;                   /* Logical Start    Sector  Number    */
int     sec_len;                /* Logical Sector   Length  (Counts)  */
char    buf[1024];              /* Work    Buffer   (1024 Bytes)      */
int     hstat;                  /* HD      Status                     */
char    *bpb_ptr;               /* BPB     Data     Pointer           */
int     l_per_p;                /* Logical Sector / Physical Sector   */


main()
{
        l_per_p = bpb_h.b_sector / P_SECTOR_SIZE;

        if(make_bpb() != OK) {  /* BPBの作成、書き込み */
                fprintf(stderr, "¥007Can not Make BPB.¥n");
        }

        if(init_fat() != OK) {  /* FATの初期化、書き込み */
                fprintf(stderr, "¥007Can not Initialize FAT.¥n");
        }

        if(init_dir() != OK) {  /* ルートディレクトリの初期化、書き込み */
                fprintf(stderr, "¥007Can not Initialize Directory.¥n");
        }

}



/*
*       Make  BPB  by  bpb_h
*
*       Return Value: OK = 0, Error = Not Zero
*/
int     make_bpb()
{
int     i,n,m;

        hstat   = OK;
        bpb_ptr = (char *)&bpb_h;
        for(i = 0;i < bpb_h.b_sector; ++i) {
                buf[i] = '¥0';  /* 書き込みセクタデータのクリア */
        }

        stno    = 0;
        sec_len = 1;

        for(i = 0; i < bpb_h.resv_sector; ++i) {
                if(i == 0) {    /* First  Reserve Sector Data */
                        n = 0;  /* セクタ0のデータセット */
```

```
                        for(m = 0;m < JMAK_SIZE; ++m) {/*メーカ名データセット*/
                                buf[n++] = *(jmak_dat+m);
                        }
                        for(m = 0;m < BPB_SIZE; ++m) {  /* ＢＰＢデータセット*/
                                buf[n++] = *(bpb_ptr+m);
                        }
                } else {        /* Other  Reserve Sector Data */
                        for(n = 0;n < (JMAK_SIZE + BPB_SIZE); ++n) {
                                buf[n] = 0;     /* セクタ１以後のデータセット*/
                        }
                }
                        /* HDD  Write */       /* セクタの書き込み */
                hstat = hwrite( stno * l_per_p,  sec_len * l_per_p,
                                                (char far *)buf );
                if(hstat != OK) {
                        break;  /* Write Error     */
                }
                stno++;         /* Increatment Write Secotr No.*/
        }

        return(hstat);
}



/*
*       Initialize  FAT  by  bpb_h
*
*       Return Value: OK = 0, Error = Not Zero
*/
int     init_fat()
{
int     i,n;

        hstat   = OK;
        for(i = 0;i < bpb_h.b_sector; ++i) {    /* Clear  Buffer  */
                buf[i] = '¥0';  /* 書き込みセクタデータのクリア */
        }

        stno    = bpb_h.resv_sector;
        sec_len = 1;

        for(i = 0; i < bpb_h.fat; ++i) {
                for(n = 0; n < bpb_h.fat_sec; ++n) {
                        if(n == 0) { /* ＦＡＴ先頭セクタデータのセット */
                                buf[0] = bpb_h.med_id;  /* FAT-ID   */
                                buf[1] = buf[2] = 0xFF; /* 予約ｺｰﾄﾞ */
                        } else {
                                buf[0] = buf[1] = buf[2] = 0; /* 未使用ｺｰﾄﾞ */
                        }
                                /* HDD  Write */       /* セクタの書き込み */
                        hstat = hwrite( stno * l_per_p,  sec_len * l_per_p,
                                                (char far *)buf );
                        if(hstat != OK) {
                                break;  /* Write Error      */
                        }
                        stno++;         /* Increatment Write Secotr No.*/
                }
        }

        return(hstat);  /* hstat: 0 = Success(OK),  Not Zero = Error */
}


/*
*       Initialize  Directory  by  bpb_h
*
*       Return Value: OK = 0, Error = Not Zero
*/
int     init_dir()
{
int     dir_sec;                /* Directory  Sector  Counts */
int     i;


        hstat   = OK;
        for(i = 0;i < bpb_h.b_sector; ++i) {
                buf[i] = '¥0';  /* 書き込みセクタデータのクリア */
        }

        stno    = bpb_h.resv_sector + bpb_h.fat_sec * bpb_h.fat;
        sec_len = 1;
        /* ディレクトリ数よりディレクトリセクタ数を求める */
        dir_sec = bpb_h.dir_no / (bpb_h.b_sector / 32);
        if(bpb_h.dir_no % (bpb_h.b_sector / 32))        ++dir_sec;

        for(i = 0;i < dir_sec; ++i) {
                /* HDD  Write */        /* セクタの書き込み */
                hstat = hwrite( stno * l_per_p,  sec_len * l_per_p,
                                                (char far *)buf );
                if(hstat != OK) {
                        break;  /* Write Error     */
                }
                stno++;         /* Increatment Write Secotr No.*/
        }

        return(hstat);
}
/*******************************************************/



/*******************************************************/
/***    BIOS  Interface  Routine                     ***/
/*******************************************************/

struct  cdb     {
        char    cmd;            /* Command */
        char    lun;            /* Logical Unit No. */
        char    st_sec_h;       /* Start Sector No.  (MSB) */
        char    st_sec_mh;
        char    st_sec_ml;
        char    st_sec_l;       /*                   (LSB) */
        char    fix_1;
        char    sec_cnt_h;      /* Sector Couts      (MSB) */
        char    sec_cnt_l;      /*                   (LSB) */
        char    fix_2;
}  cdb_rec = {                  /*  Read  Extended  Command */
        0x28,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
}, cdb_wec = {                  /*  Write Extended  Command */
        0x2A,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};



/*
*       SCSI  HD   Read(Physical  Sector)
*       Argument:
*               unsigned long   p_st_secno;     Start  Sector Number ( 0 - N)
*               unsigned int    p_sec_len;      Sector Length (Counts)
*               char     far    *buf;           Data   Buffer Address
*       Return Value:
*               OK = 0  ,       Error = Not Zero
*/
int     hread( p_st_secno, p_sec_len, buf)
unsigned long   p_st_secno;     /* Start  Sector Number ( 0 - N) */
unsigned int    p_sec_len;      /* Sector Length (Counts)        */
char    far     *buf;           /* Data   Buffer Address         */
{
int     HDproc();
void    d_swap();
int     status;

        d_swap((char *)&cdb_rec.st_sec_h ,  (char *)&p_st_secno, sizeof(long) );
        d_swap((char *)&cdb_rec.sec_cnt_h,  (char *)&p_sec_len , sizeof(int)  );
        status = HDproc( cdb_rec.cmd, (char far *)&cdb_rec, sizeof(cdb_rec),
                                buf, p_sec_len * P_SECTOR_SIZE );
        return(status);
}



/*
*       SCSI  HD   Write(Physical  Sector)
*       Argument:
*               unsigned long   p_st_secno;     Start Sector Number( 0 - N)
*               unsigned int    p_sec_len;      Sector Length (Counts)
*               char     far    *buf;           Data   Buffer Address
```

〈リスト 2〉 FORMATHD.C(つづき)

```
*       Return Value:
*               OK = 0  ,       Error = Not Zero
*/
int             hwrite( p_st_secno, p_sec_len, buf)
unsigned long   p_st_secno;     /* Start  Sector Number ( 0 - N) */
unsigned int    p_sec_len;      /* Sector Length (Counts)        */
char     far    *buf;           /* Data   Buffer Address         */
{
int             HDproc();
void            d_swap();
int             status;

                d_swap((char *)&cdb_wec.st_sec_h , (char *)&p_st_secno, sizeof(long) );
                d_swap((char *)&cdb_wec.sec_cnt_h, (char *)&p_sec_len , sizeof(int)  );
                status = HDproc( cdb_wec.cmd, (char far *)&cdb_wec, sizeof(cdb_wec),
                                buf, p_sec_len * P_SECTOR_SIZE );
                return(status);
}



/*
*       DATA    SWAP  Sub-Routine
*       Argument:
*               char    *dptr;          Destination Pointer
*               char    *sptr;          Source      Pointer
*               int     len;            Swap        Length(Counts)
*       Return Value:
*               Nothing
*/
void            d_swap( dptr, sptr, len)
char            *dptr, *sptr;
int             len;
{
int             i;

                for(i = 0; i < len; ++i) {
                        *(dptr + i) = *(sptr + (len - 1) - i);
                }
}

/*******************************************************/

```

〈リスト 3〉 BIOSHD.ASM

```
;
                PAGE    132,66
_TEXT           SEGMENT WORD PUBLIC     'CODE'
                ASSUME  CS:_TEXT
                PUBLIC  _HDproc                         ;SCSI DISK ACCESS ROUTINE
;
;       HD INTERRUPT PROCESS
;       HDproc(ah, ds:bx, dx, es:di, cx)
;               ah      SCSI    Command
;               ds:bx   CDB     Pointer
;               dx      CDB     Length
;               es:di   DATA    Buffer Pointer
;               cx      DATA    Length
;
_HDproc         proc    near
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI
                PUSH    DI
                PUSH    DS
                PUSH    ES
                PUSH    BP
                MOV     BP,SP
;
                MOV     AX,18[BP]
                LDS     BX,20[BP]
                MOV     DX,24[BP]
                LES     DI,26[BP]
                MOV     CX,30[BP]
                INT     40H             ;uPD72111 DISK ACCESS(Adr=100H)
                                        ;AL:SCSI STATUS (Good = 0, Else )
                                        ;AH:SCSI FLAG-1 (OK   = 0, Else Error)
;
                POP     BP
                POP     ES
                POP     DS
                POP     DI
                POP     SI
                POP     DX
                POP     CX
                POP     BX
                RET
_HDproc         endp
_TEXT           ENDS
                END
```

〈リスト 4〉 SCSUB.ASM

```
;       デバイス・ドライバ
;       アッセンブラ部分
;
                page 50,128
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
public  _scsi_bios                      ;物理ＢＩＯＳをコールするルーチン
public  _io_init                        ;Ｉ／Ｏのイニシャライズ処理
;
extrn   _d_entry:near                   ;エントリ処理
extrn   scbios_init:near                ;物理ＢＩＯＳ初期化
_TEXT ENDS
;
_DATA SEGMENT WORD PUBLIC 'DATA'
;
extrn   _cha:word                       ;コマンド先頭アドレス
extrn   _dha_off:word                   ;データ先頭アドレス・オフセット
extrn   _dha_seg:word                   ;データ先頭アドレス・セグメント
extrn   _tcc:byte                       ;トランスファ・コマンド・カウンタ
extrn   _tdc:word                       ;トランスファ・データ・カウンタ
_DATA ENDS
;
_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS
;
_PEND SEGMENT WORD PUBLIC 'PEND'
_PEND ENDS
;
DGROUP GROUP _TEXT,_DATA,_BSS,_PEND
        ASSUME  CS:_TEXT,DS:DGROUP,SS:DGROUP,ES:DGROUP
;
;
_DATA SEGMENT WORD PUBLIC 'DATA'
;
_stte           db      ?               ;ステータスコード
_scsif          db      ?               ;ＳＣＳＩリザルトフラグ
                                        ;b0:FIFOｵｰﾊﾞﾗﾝ/ｱﾝﾀﾞﾗﾝ･ｴﾗｰ･ﾌﾗｸﾞ
                                        ;b1:REQ/ACKﾀｲﾑｱｳﾄ･ｴﾗｰ･ﾌﾗｸﾞ
                                        ;b2:ﾌｪｰｽﾞ･ｴﾗｰ･ﾌﾗｸﾞ
                                        ;b3:ﾀｲﾑｱｳﾄ･ﾌﾗｸﾞ
                                        ;b4:SCSIﾊﾞｽ･ﾊﾟﾘﾃｨ･ｴﾗｰ･ﾌﾗｸﾞ
                                        ;b5:メッセージリジェクト
                                        ;b6:SCSIﾘｾｯﾄ･ﾌﾗｸﾞ
                                        ;b7:ﾃﾞｨｽｺﾈｸﾄ･ﾌﾗｸﾞ
_packet         dd      ?               ;コマンド・パケットのポインタ領域
stacksave       dd      ?               ;スタック・ポインタのセーブ領域
```

〈リスト 4〉 SCSUB.ASM(つづき)

```
_prog_off       dw      ?               ;プログラム終了アドレスのセーブ領域
_prog_seg       dw      ?
;
hyouji  db      'ＳＣＳＩデバイス・ドライバ　Ｖｅｒ　１．１を組み込みました。
        db      0dh,0ah,0dh,0ah,'$'
;
_work_buf       db      1024*3 dup(?)   ;Work  Buffer 2 -> 3
stacktop        label   byte            ;スタック先頭アドレス
_DATA ENDS
;
_PEND SEGMENT WORD PUBLIC 'PEND'
progend         label   word            ;プログラム終了アドレス
_PEND ENDS
;
public  _packet                         ;コマンド・パケットのポインタ領域
public  _prog_off,_prog_seg             ;プログラム終了アドレスのセーブ領域
public  _work_buf                       ;BPB  読み込みバッファ
public  _stte                           ;ステータスコード
public  _scsif                          ;ＳＣＳＩリザルトフラグ
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
;
;************************************************************************
;*                      デバイス・ヘッダの構成                          *
;************************************************************************
;
header  label   word
        dd      -1                      ;次のデバイスへのポインタ
        dw      2000h                   ;デバイスの属性
        dw      strategy_r              ;ストラテジ・エントリ・ポイント
        dw      interrupt_r             ;割り込みエントリ・ポイント
        dd      1                       ;ユニット数
;
;********************************************************************************
;*                      ストラテジ・ルーチン                                    *
;*                                                      入力：bx,es             *
;*コマンド・パケットの置かれているアドレスを格納する。  出力：_packet           *
;********************************************************************************
;
strategy_r      proc    far
;
        mov     word ptr cs:_packet,bx
        mov     word ptr cs:_packet+2,es
        ret
;
strategy_r      endp
;********************************************************************************
;*                      割り込みルーチン                                        *
;*                                                      入力：なし              *
;*ＳＣＳＩデバイス・ドライバの入口と出口。この処理の    出力：_prog_off,_prog_seg *
;*中でエントリ処理をＣＡＬＬする。                                              *
;********************************************************************************
;
interrupt_r     proc    far
;
        push    ax                      ;レジスタ退避
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    bp
        push    ds
        push    es
;
        mov     ax,cs                           ;ｄｓ　＝　ｃｓ
        mov     ds,ax
        mov     word ptr stacksave,sp           ;スタック・ポインタの退避
        mov     word ptr stacksave+2,ss
;
        cli
        mov     ss,ax                           ;ローカル・スタックの確保
        mov     sp,offset DGROUP:stacktop
        sti
;
        mov     _prog_off,offset DGROUP:progend ;プログラムの終了アドレスを_prog_seg,_prog_offにセット
        mov     _prog_seg,cs
        call    _d_entry                        ;エントリ処理
;
        cli
        mov     sp,word ptr stacksave           ;スタック・ポインタの復帰
        mov     ss,word ptr stacksave+2
        sti
;
        pop     es                              ;レジスタ復帰
        pop     ds
        pop     bp
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
;
interrupt_r     endp
;****************************************************************************************
;*                      Ｉ／Ｏのイニシャライズ処理                                      *
;*                                                                      入力：なし      *
;*“ＳＣＳＩデバイス・ドライバを組み込みました。”と表示する。          出力：なし      *
;*割り込みベクタのテーブルのセット。                                                    *
;*μＰＤ７２１１１のイニシャライズ。                                                    *
;*ＳＣＳＩリセット。                                                                    *
;****************************************************************************************
;
_io_init        proc    near
;
        push    bp
        push    si
        push    di
        call    scbios_init                     ;物理ＢＩＯＳ初期化
        mov     dx,offset DGROUP:hyouji         ;メッセージを表示する
        mov     ah,09h
        int     21h
        pop     di
        pop     si
        pop     bp
        ret
_io_init        endp
;
;  ｃ から、物理ＢＩＯＳをコールするためのサブルーチン
;
_scsi_bios      proc
        push    di
        push    es
;
        mov     bx,_cha                 ;コマンド先頭アドレス・オフセット
        mov     dh,0
        mov     dl,_tcc                 ;トランスファ・コマンド・カウンタ
        mov     cx,_tdc                 ;トランスファ・データ・カウンタ
        mov     di,_dha_off             ;データ先頭アドレス・オフセット
        mov     es,_dha_seg             ;データ先頭アドレス・セグメント
        int     40h                     ;Software-Interrupt SCSI BIOS
        mov     _stte,al                ;al = SCSI Status
        mov     _scsif,ah               ;ah = return status
        mov     al,ah                   ;ax as result
        pop     es
        pop     di
        ret
_scsi_bios      endp
;
;
;
;MS-C  Routine
public  __acrtused                      ; trick to force in startup
;
__acrtused      proc
        ret
__acrtused      endp

;
;
_TEXT ENDS
;
;
;
END
```

〈リスト 5〉 SCDV.C

```

/* μＰＤ７２１１１デバイス・ドライバ（ＳＣＤＶ．Ｃ） */

/* #include <dos.h> */

    /* コマンド・パケット・ステータス情報 */

#define     ST_ERROR        0x8000  /* ERRORビット */
#define     ST_DONE         0x0100  /* DONEビット */
#define     ST_ERR_NOTRDY        2  /* ドライブの準備ができていない */
#define     ST_ERR_BADCMD        3  /* コマンド番号が間違っている */
#define     ST_ERR_CRC           4  /* ＣＲＣエラー */
#define     ST_ERR_SEEK          6  /* シーク・エラー */
#define     ST_ERR_SECTOR        8  /* セクタが見つかりません */
#define     ST_ERR_WRITE        10  /* 書き込みエラー */
#define     ST_ERR_GENERAL      12  /* その他のエラー */

    /* センス・コード情報 */

#define     R_R     0x17    /* ﾘｶﾊﾞｰ･ﾘｰﾄﾞ･ﾃﾞｰﾀ･ｳｨｽﾞ･ﾘｰﾄﾞ･ﾘﾄﾗｲ */
#define     ECC     0x18    /* ﾘｶﾊﾞｰ･ﾚｰﾄﾞ･ﾃﾞｰﾀ･ｳｨｽﾞECC */
#define     U_RE    0x11    /* ｱﾝﾘｶﾊﾞｰ･ﾘｰﾄﾞ･ｴﾗｰ */
#define     N_RF    0x14    /* ﾉｰ･ﾚｺｰﾄﾞ･ﾌｧｲﾝﾄﾞ */
#define     N_SC    0x02    /* ﾉｰ･ｼｰｸ･ｺﾝﾌﾟﾘｰﾄ */
#define     SE      0x15    /* ｼｰｸ･ｴﾗｰ */
#define     W_F     0x03    /* ﾗｲﾄ･ﾌｫﾙﾄ */
#define     N_TOF   0x06    /* ﾉｰ･ﾄﾗｯｸ 0 ﾌｧｲﾝﾄﾞ */
#define     P_DF    0x42    /* ﾊﾟﾜｰ･ｵﾝ･ﾀﾞｲｱｸﾞﾉｽﾃｨｸ･ﾌｪｲﾗｰ */
#define     P_O     0x29    /* ﾊﾟﾜｰ･ｵﾝ,ﾘｾｯﾄ,ﾊﾞｽ･ﾃﾞﾊﾞｲｽ･ﾁｪﾝｼﾞ */
#define     N_R     0x04    /* ﾉｯﾄ･ﾚﾃﾞｨ */

    /* コマンド，データのバイト数 */

#define     RSC_BC      0x06    /* ﾘｸｴｽﾄ･ｾﾝｽ･ｺﾏﾝﾄﾞのバイト数 */
#define     RSD_BC      0x16    /* ﾘｸｴｽﾄ･ｾﾝｽ･ﾃﾞｰﾀのバイト数 */
#define     REC_BC      0x0A    /* ﾘｰﾄﾞ･ｴｸｽﾃﾝﾃﾞｨﾄﾞ･ｺﾏﾝﾄﾞのバイト数 */
#define     WEC_BC      0x0A    /* ﾗｲﾄ･ｴｸｽﾃﾝﾃﾞｨﾄﾞ･ｺﾏﾝﾄﾞのバイト数 */
#define     WAVC_BC     0x0A    /* ﾗｲﾄ･ｱﾝﾄﾞ･ﾍﾞﾘﾌｧｲ･ｺﾏﾝﾄﾞのバイト数 */
#define     TURC_BC     0x06    /* ﾃｽﾄ･ﾕﾆｯﾄ･ﾚﾃﾞｨ･ｺﾏﾝﾄﾞのバイト数 */
#define     SSUCI_BC    0x06    /* ｽﾀｰﾄ/ｽﾄｯﾌﾟ･ﾕﾆｯﾄ･ｺﾏﾝﾄﾞのバイト数 */
#define     MSC_BC      0x06    /* ﾓｰﾄﾞ･ｾﾝｽ･ｺﾏﾝﾄﾞのバイト数 */
#define     MSD_BC      0x40    /* ﾓｰﾄﾞ･ｾﾝｽ･ﾃﾞｰﾀのバイト数 */
#define     MSLC_BC     0x06    /* ﾓｰﾄﾞ･ｾﾚｸﾄ･ｺﾏﾝﾄﾞのバイト数 */
#define     MSLD_BC     0x40    /* ﾓｰﾄﾞ･ｾﾚｸﾄ･ﾃﾞｰﾀのバイト数 */


    /* コマンド・パケットの構造体 */

typedef struct {
        char command_length;            /* コマンド・パケットのバイト数 */
        char unit_number;               /* 論理装置コード */
        char cmd;                       /* Ｉ／Ｏリクエスト・コマンド・コード */
        int  command_status;            /* ステータス */
        long resv1,resv2;               /* 予約域　（８バイト） */

        union {

            /* build bpbのコマンド・パケット */

            struct {
                char mediab;            /* ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ */
                int  transb_off;        /* 転送アドレス */
                int  transb_seg;
                struct bpb_st *bpb_off; /* ＢＰＢに対するポインタ */
                char *bpb_seg;
                } bb;

            /* リード，ライトのコマンド・パケット */

            struct {
                char mediarw;           /* ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ */
                char *transrw_off;      /* 転送アドレス */
                char *transrw_seg;
                char sector_count_l;    /* セクタ・カウント */
                char sector_count_m;
                char start_sector_l;    /* 開始セクタ番号 */
                char start_sector_m;
                } rw;

            /* メディア・チェックのコマンド・パケット */

            struct {
                char mediamc;           /* ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ */
                int  retpr;             /* 返す値 */
                } mc;

            /* イニシャライズのコマンド・パケット */

            struct {
                char unit;              /* ユニット数 */
                char *end_off;          /* ブレーク・アドレス */
                char *end_seg;
                struct bpb_st **bpb_di_off; /* ＢＰＢの配列ポインタ */
                char *bpb_di_seg;
                } in;

        } up;

    } command_packet;

    /* ＢＩＯＳ　コマンド・パケット */
typedef struct {
        int  cmd;                       /* ＢＩＯＳリクエスト・コマンド・コード */
        char sector_count_l;            /* セクタ・カウント */
        char sector_count_m;
        char *transrw_off;              /* 転送アドレス */
        char *transrw_seg;
        char start_sector_l;            /* 開始セクタ番号 */
        char start_sector_hl;
        char start_sector_m;
        char start_sector_hm;
} cmd_bios_packet;


    /* ＢＰＢの構造体 */

struct  bpb_st {
        int  b_sector;          /* １セクタ当りのバイト数 */
        char sector_c;          /* １クラスタ当りのセクタ数 */
        int  resv_sector;       /* 予約セクタ数 */
        char fat;               /* ＦＡＴの数 */
        int  dir_no;            /* ディレクトリ数 */
        int  tol_sec;           /* 論理イメージのセクタ数 */
        char med_id;            /* メディア・ディスクリプタ・バイト */
        int  fat_sec;           /* ＦＡＴセクタ数 */
        } bpb_h = {
            1024,
              16,
               1,
               2,
            3072,
           39600,
            0xF8,
               4
        };

    /* エクステンディド・コマンドの構造体 */

struct extented_command {
                char oc[2];
                char start_scn_hm;      /* 開始セクタ番号(High-MSB) */
                char start_scn_m;       /* 開始セクタ番号(MSB)      */
                char start_scn_hl;      /* 開始セクタ番号(High-LSB) */
                char start_scn_l;       /* 開始セクタ番号(LSB)      */
                char fix_l;
                char trans_sector_m;    /* 転送セクタ数(MSB)  */
                char trans_sector_l;    /* 転送セクタ数(LSB)  */
                char fix_2;
                }

        /* リード・エクステンディド・コマンド */

        cha_rec = {
                0x28,0x00,0x00,0x00,
                0x00,
                0x00,
                0x00,
```

```
                        0x00,
                        0x00,
                        0x00
                          },

            /* ライト・エクステンディド・コマンド */

            cha_wec = {
                     0x2A,0x00,0x00,0x00,
                     0x00,
                     0x00,
                     0x00,
                     0x00,
                     0x00,
                     0x00,
                     0x00
                   },

            /* ライト・アンド・ベリファイ・コマンド */

            cha_wavc = {
                      0x2E,0x00,0x00,0x00,
                      0x00,
                      0x00,
                      0x00,
                      0x00,
                      0x00,
                      0x00,
                      0x00
                    };

      /* コマンド，データの配列 */

/* リクエスト・センス・コマンド */
char cha_rsc[6] = { 0x03,0x00,0x00,0x00,0x16,0x00 };

/* リクエスト・センス・データ */
char dha_rsd[22];

/* テスト・ユニット・レディ・コマンド */
char cha_turc[6] = { 0x00,0x00,0x00,0x00,0x00,0x00 };

/* スタート／ストップ・ユニット・コマンド（スタート） */
char cha_ssuc1[6] = { 0x1B,0x00,0x00,0x00,0x01,0x00 };

/* モード・センス・コマンド */
char cha_msc[6] = { 0x1A,0x00,0x3F,0x00,0x40,0x00 };

/* モード・センス・データ */
char dha_msd[64];

/* モード・センス・コンパイル・データ */
char dha_msd_c[64] = { 0x3F,0x00,0x00,0x08,
                       0x00,0x01,0x46,0xB8,0x00,0x00,0x02,0x00,
                       0x01,0x06,0x00,0x08,0x00,0x00,0x00,0x00,
                       0x03,0x16,0x00,0x04,0x00,0x04,0x00,0x00,
                       0x00,0x14,0x00,0x22,0x02,0x00,0x00,0x01,
                       0x00,0x12,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x04,0x12,0x00,0x02,0x67,0x04,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00 };
/* OLD
char dha_msd_c[64] = { 0x3F,0x00,0x00,0x08,
                       0x00,0x00,0xA5,0x00,0x00,0x00,0x04,0x00,
                       0x81,0x06,0x24,0x10,0x08,0x00,0x00,0xFF,
                       0x83,0x16,0x00,0x08,0x00,0x10,0x00,0x00,
                       0x00,0x50,0x00,0x0C,0x00,0x00,0x00,0x01,
                       0x00,0x00,0x00,0x00,0x80,0x00,0x00,0x00,
                       0x04,0x12,0x00,0x01,0xB8,0x08,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00 };
*/
/* モード・セレクト・コマンド */
char cha_mslc[6] = { 0x15,0x01,0x00,0x00,0x40,0x00 };

/* モード・セレクト・データ */
char dha_msld[64] = { 0x00,0x00,0x00,0x08,
                      0x00,0x01,0x46,0xB8,0x00,0x00,0x02,0x00,
                      0x01,0x06,0x00,0x08,0x00,0x00,0x00,0x00,
                      0x03,0x16,0x00,0x04,0x00,0x04,0x00,0x00,
                      0x00,0x14,0x00,0x22,0x02,0x00,0x00,0x01,
                      0x00,0x12,0x00,0x00,0x00,0x00,0x00,0x00,
                      0x04,0x12,0x00,0x02,0x67,0x04,0x00,0x00,
                      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x00 };
/* OLD
char dha_msld[64] = { 0x00,0x00,0x00,0x08,
                      0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x00,
                      0x01,0x06,0x24,0x10,0x08,0x00,0x00,0x00,
                      0x03,0x16,0x00,0x08,0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                      0x04,0x12,0x00,0x01,0xB8,0x08,0x00,0x00,
                      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x00 };
*/

    /* ＢＩＯＳ：ｐｈａｓｅに出力する変数 */

union {                                  /* コマンド先頭アドレス */
        struct extented_command *cha_st;
        char *cha_c;
      }cha;

char *dha_off;                           /* データ先頭アドレス・オフセット */
char *dha_seg;                           /* データ先頭アドレス・セグメント */
char tcc;                                /* トランスファ・コマンド・カウンタ*/
int  tdc;                                /* トランスファ・データ・カウンタ */
/****/
cmd_bios_packet bios_packet;             /* ＢＩＯＳ・コマンド・パケット */
/****/
char tdat[4];                            /* Temporary Convert Data        */
int  bscnt = 1024 / 512;                 /* Block  Size  Counts           */
                                         /* Logical Sector / Physical Sector */
/****/

     /* エクスタン変数 */

extern char scsif;                   /* ＳＣＳＩフラグ      (phase) */
extern char stte;                    /* ステータス領域  (phase) */

extern command_packet far *packet;   /* コマンド・パケットのポインタ (interrupt_r) */
extern char *prog_off;               /* プログラム・エンドのオフセット値 (interrupt_r) */
extern char *prog_seg;               /* プログラム・エンドのセグメント値 (interrupt_r) */

extern void io_init( );              /* Ｉ／Ｏのイニシャライズ処理 */

extern void scsi_bios( );            /* ＳＣＳＩ  ＢＩＯＳ（Ｐｈａｓｅ）*/
                                     /* μＰＤ７２１１１制御処理        */

extern char work_buf;                /* BPB  Read  Buffer   */

struct bpb_st *bpbpoint = &bpb_h;    /* ＢＰＢの配列ポインタ */

void d_entry( );                 /* エントリ処理 */
void rsc( );                     /* リクエスト・センス・コマンド設定処理 */
void turc( );                    /* テスト・ユニット・レディ・コマンド設定処理 */
void ssuc_1( );                  /* スタート／ストップ・ユニット・コマンド設定処理 */
void msc( );                     /* モード・センス・コマンド設定処理 */
void mslc( );                    /* モード・セレクト・コマンド設定処理 */
void rec( );                     /* リード・コマンド設定処理 */
void wec( );                     /* ライト・コマンド設定処理 */
void wavc( );                    /* ライト・アンド・ベリファイ設定処理 */
int  stjudg( );                  /* ステータス判断処理 */
char ready_h( );                 /* ＨＤＤレディ処理 */
/****/
int bios_rsc( );                 /* ＢＩＯＳリクエスト・センス・コマンド処理 */
int bios_msc( );                 /* ＢＩＯＳモード・センス・コマンド処理 */
int bios_ssuc_1( );              /* ＢＩＯＳスタート／ストップ・ユニット・コマンド処理*/
int bios_turc( );                /* ＢＩＯＳテスト・ユニット・レディ・コマンド処理 */
int bios_mslc( );                /* ＢＩＯＳモード・セレクト・コマンド処理 */
int bios_rec( );                 /* ＢＩＯＳリード・コマンド処理          */
int bios_wec( );                 /* ＢＩＯＳライト・コマンド処理          */
int bios_wavc( );                /* ＢＩＯＳライト・アンド・ベリファイ処理 */
int scsi_read( );                /* ＳＣＳＩリード・コマンド処理          */
int scsi_write( );               /* ＳＣＳＩライト・コマンド処理          */
int scsi_writev( );              /* ＳＣＳＩライト・アンド・ベリファイ処理 */
void bios_packet_set( );         /* ＢＩＯＳパケットデータ設定処理        */
int  bpb_input( );               /* Read Top  Sector(Physical)  */
void bpb_set( );                 /* Set  BPB  Set               */
```

〈リスト 5〉SCDV.C(つづき)

```
void bios_bpb_rec( );            /* Set  BPB  BIOS Read Command */
/****/
void init( ),media_check( ),build_bpb( ),input( ),
     output( ),output_v( ),error_io( );

    /*  I／Oリクエスト・テーブル  */

void (*func[10])( ) = {
            init,        /*  イニシャライズ処理                0  */
            media_check, /*  メディア・チェック処理            1  */
            build_bpb,   /*  ＢＵＩＬＤ  ＢＰＢ処理            2  */
            error_io,    /*  Ｉ／Ｏリクエスト・エラー処理      3  */
            input,       /*  インプット処理                    4  */
            error_io,    /*  Ｉ／Ｏリクエスト・エラー処理      5  */
            error_io,    /*  Ｉ／Ｏリクエスト・エラー処理      6  */
            error_io,    /*  Ｉ／Ｏリクエスト・エラー処理      7  */
            output,      /*  アウトプット処理                  8  */
            output_v     /*  アウトプット・ウィズ・ベリファイ処理  9  */
            };


    /************************************************************************************/
    /*                            エントリ処理                                          */
    /*                                                                                  */
    /*Ｉ／Ｏリクエスト・コマンド・コードによって処理をＣＡＬＬする。 入力：packet->cmd    */
    /*Ｉ／Ｏリクエスト・コードが０ＡＨ以下のものが入力してくると    出力：packet->command_status */
    /*Ｉ／Ｏリクエスト・コードが間違っていることをＭＳ－ＤＯＳに渡す。                 */
    /************************************************************************************/

void d_entry( )
    {
        if(packet->cmd > 9)
            {
                packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_BADCMD);  /*  Ｉ／Ｏリクエスト・コマンド・
コードが間違っている  */
            }
        else
            {
                (*func[packet->cmd])( );                               /*   処理に分岐する  */
            }
        return;
    }


    /************************************************************************************/
    /*                        イニシャライズ処理                                        */
    /*                                                                                  */
    /*μＰＤ７２１１１のイニシャライズ及びＨＤＤのイニシャライズを   入力：prog_off,prog_seg */
    /*行なう。                                                       出力：packet        */
    /************************************************************************************/

void init( )
    {
        int  status;
        char c;

        io_init( );                                        /*  Ｉ／Ｏのイニシャライズ処理  */
        if(scsif!=0)                                       /*  エラー発生か？  */
            {
             packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);/* その他のエラー */
             return;
            }
        if(ready_h( )!=0)                                  /*  レディ状態か？ */
            {
                return;
            }
        bios_msc( );                                       /*  ＢＩＯＳモード・センス・コマンド処理*/
        if((status = stjudg( ))!=0)                        /*  コマンド正常終了か？ */
            {
                packet->command_status = status;           /* エラー */
                return;
            }
        c = 0;

        /* モード・センス・データと希望のモード・センス・データの比較処理  */
        while (c<64)
            {
                if(dha_msd[c] == dha_msd_c[c])
                    {
                        c++;
                    }
                else
                    {
                        bios_mslc( );                             /*  ＢＩＯＳモード・セレクト・コマンド処理*/
                        if((status = stjudg( )) == 0)             /*  コマンド正常終了か？ */
                            {
                                break;
                            }
                        else
                            {
                                packet->command_status = status;  /* エラー */
                                return;
                            }
                    }
            }
        packet->up.in.unit = 1;                                   /*  ユニット数１をセット  */
        packet->up.in.end_off = prog_off;                         /*  ブレーク・アドレスをセット  */
        packet->up.in.end_seg = prog_seg;

        packet->up.in.bpb_di_off = &bpbpoint;                     /*  ＢＰＢの配列へのポインタをセット  */
        packet->up.in.bpb_di_seg = prog_seg;
        packet->command_status = ST_DONE;                         /*  ＤＯＮＥビットをセット  */
        return;
    }


    /************************************************************************************/
    /*                              メディア・チェック処理                              */
    /*                                                                                  */
    /*ＭＳ－ＤＯＳに“メディアが変更されていない”ことを返す。       入力：なし           */
    /*                                                               出力：packet         */
    /************************************************************************************/

void media_check( )
    {
        packet->up.mc.retpr = 1;              /*  返す値に１をセット  */
        packet->command_status = ST_DONE;     /*  ＤＯＮＥビットをセット  */
        return;
    }


    /************************************************************************************/
    /*                              ＢＵＩＬＤ  ＢＰＢ処理                              */
    /*                                                                                  */
    /*ＳＣＳＩデバイス・ドライバ内に用意してあるＢＰＢのポインタを   入力：prog_seg       */
    /*ＭＳ－ＤＯＳに返す。                                           出力：packet         */
    /************************************************************************************/

void build_bpb( )
    {
        if(bpb_input() == 0) {
                 bpb_set();
        }
        packet->up.bb.mediab = bpb_h.med_id;  /*  ＤＰＢへのメディア・ディスクリプタ・バイトをセット  */
        packet->up.bb.bpb_off = &bpb_h;       /*  ＢＰＢに対するポインタをセット  */
        packet->up.bb.bpb_seg = prog_seg;
        packet->command_status = ST_DONE;     /*  ＤＯＮＥビットをセット  */
        return;
    }


    /************************************************************************************/
    /*                          Ｉ／Ｏリクエスト・エラー処理                            */
    /*                                                                                  */
    /*ＭＳ－ＤＯＳに“Ｉ／Ｏリクエスト・コマンド・コードが間違って   入力：なし           */
    /*いる”を返す。                                                 出力：packet         */
    /************************************************************************************/

void error_io( )
    {
        packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_BADCMD); /*  Ｉ／Ｏリクエスト・
コマンド・コードが間違っている*/
        return;
    }
```

〈リスト5〉SCDV.C(つづき)

```
/*****************************************************************************/
/*                         インプット処理                                    */
/*                                                                           */
/*リード・エクステンディド・コマンドを送信して,          入力：なし          */
/*ハード・ディスク・ドライブからデータを                  出力：packet        */
/*リードする。                                                               */
/*****************************************************************************/

void input( )
    {
    int status;

        bios_packet_set( );                 /* リード・コマンド設定処理 */
        if((status = scsi_read( )) == 0)    /* ＢＩＯＳリード・コマンド処理 */
                                            /* コマンド正常終了か？     */
            {
            packet->command_status = ST_DONE;   /* ＤＯＮＥビットをセット  */
            } else {
            packet->command_status = status;    /* エラー */
            }
        return;
    }


/*****************************************************************************/
/*                         アウトプット処理                                  */
/*                                                                           */
/*ライト・エクステンディド・コマンドを送信して,          入力：なし          */
/*ハード・ディスク・ドライブにデータをライトする。        出力：packet        */
/*****************************************************************************/

void output( )
    {
    int status;

        bios_packet_set( );                 /* ライト・コマンド設定処理 */
        if((status = scsi_write( )) == 0)   /* ＢＩＯＳライト・コマンド処理 */
                                            /* コマンド正常終了か？     */
            {
            packet->command_status = ST_DONE;   /* ＤＯＮＥビットをセット  */
            } else {
            packet->command_status = status;    /* エラー */
            }
        return;
    }


/*****************************************************************************/
/*                  アウトプット・ウィズ・ベリファイ処理                     */
/*                                                                           */
/*ライト・アンド・ベリファイ・コマンドを送信して，ハード・ディス  入力：なし   */
/*ク・ドライブ・にデータをライトした後データをベリファイする。    出力：packet */
/*****************************************************************************/

void output_v( )
    {
    int status;

        bios_packet_set( );                 /* ライト・アンド・ベリファイ・コマンド設定処理 */
        if((status = scsi_writev( )) == 0)  /* ＢＩＯＳライト・アンド・ベリファイ処理 */
                                            /* コマンド正常終了か？ */
            {
            packet->command_status = ST_DONE;   /* ＤＯＮＥビットをセット */
            } else {
            packet->command_status = status;    /* エラー */
            }
        return;
    }


/*****************************************************************************/
/*                         ステータス判断処理                                */
/*                                                                           */
/*μＰＤ７２１１１制御処理の結果（ｓｃｓｉｆ１，ｓｃｓｉｆ２，  入力：scsif, stte */
/*ｓｔｔｅ）によってエラー時は１，正常時は０を上位関数に戻す。  出力：status     */
/*****************************************************************************/

int stjudg( )
    {
    int status;

        if(scsif!=0)                                        /* エラー発生か？ */
            {
            return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);        /* その他のエラー */
            }
        if(stte!=0)                                         /* グッド・コンディションか？ */
            {
            bios_rsc( );                                /* モード・センス・コマンド設定処理 */
            if(scsif!=0)                                    /* エラー発生か？ */
                {
                return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);    /* その他のエラー */
                }
            if(stte!=0)                                     /* グッド・コンディションか？ */
                {
                return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);    /* その他のエラー */
                }

        /* センス・コードに対する処理 */
            switch(dha_rsd[12])
                {
                case  R_R:                                  /* ﾘｶﾊﾞｰ・ﾘｰﾄﾞ・ﾃﾞｰﾀ・ｳｨｽﾞ・ﾘｰﾄﾞ・ﾘﾄﾗｲ */
                case  ECC:                                  /* ﾘｶﾊﾞｰ・ﾘｰﾄﾞ・ﾃﾞｰﾀ・ｳｨｽﾞ ECC */
                        break;
                case U_RE:                                  /* ｱﾝﾘｶﾊﾞｰ・ﾘｰﾄﾞ・ｴﾗｰ */
                        return(ST_ERROR|ST_DONE|ST_ERR_CRC);    /* ＣＲＣエラー */
                case N_RF:                                  /* ﾉｰ・ﾚｺｰﾄﾞ・ﾌｧｲﾝﾄﾞ */
                        return(ST_ERROR|ST_DONE|ST_ERR_SECTOR); /* セクタが見つかりません */
                case N_SC:                                  /* ﾉｰ・ｼｰｸ・ｺﾝﾌﾟﾘｰﾄ */
                case   SE:                                  /* ｼｰｸ・ｴﾗｰ */
                        return(ST_ERROR|ST_DONE|ST_ERR_SEEK);   /* シーク・エラー */
                case  W_F:                                  /* ﾗｲﾄ・ﾌｫﾙﾄ */
                        return(ST_ERROR|ST_DONE|ST_ERR_WRITE);  /* 書き込みエラー */
                case  N_R:                                  /* ﾉｯﾄ・ﾚﾃﾞｨ */
                        bios_ssuc_1( );                     /* ｽﾀｰﾄ/ｽﾄｯﾌﾟ・ﾕﾆｯﾄ・ｺﾏﾝﾄﾞ 設定処理 */
                        if((status = stjudg( ))==0)         /* 再帰 */
                            {
                            return(0);
                            }
                        else
                            {
                            return(status);
                            }
                case  P_O:                                  /* ﾊﾟﾜｰ・ｵﾝ, ﾘｾｯﾄ, ﾊﾞｽ・ﾃﾞﾊﾞｲｽ・ﾁｪﾝｼﾞ */
                        bios_mslc( );                       /* モード・セレクト・コマンド処理 */
                        if((status = stjudg( ))==0)         /* 再帰 */
                            {
                            return(0);
                            }
                        else
                            {
                            return(status);
                            }
                default  :
                        return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);    /* その他のエラー */
                }
            }
        return(0);
    }


/*****************************************************************************/
/*                          ＨＤＤレディ処理                                 */
/*                                                                           */
/*（１）ＨＤＤがレディ状態であるか，テスト・ユニット・レディ  入力：scsif, stte */
/*     ・コマンドを送信し，確認する。                          出力：packet      */
/*（２）レディ状態でなければ，スタート・ストップ・コマンドを                  */
/*     送信する。                                                            */
/*（１），（２）を繰り返す。                                                 */
/*  この関数を実行した結果，正常時は０，エラー時は１を上位                   */
/* 関数に戻す。                                                              */
/*****************************************************************************/

char ready_h( )
    {
        bios_turc( );                                       /* ﾃｽﾄ・ﾕﾆｯﾄ・ﾚﾃﾞｨ・ｺﾏﾝﾄﾞ 処理 */
        if(scsif!=0)                                        /* エラー発生か？ */
```

〈リスト5〉SCDV.C(つづき)

```
            {
            packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);        /* その他のエラー */
            return(1);
            }
        if(stte!=0)                                                    /* グッド・コンディションか? */
            {
            bios_rsc( );                                               /* モード・センス・コマンド処理 */
            if(scsif!=0)                                               /* エラー発生か? */
                {
                packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);    /* その他のエラー */
                return(1);
                }
            if(stte!=0)                                                /* グッド・コンディションか? */
                {
                packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);    /* その他のエラー */
                return(1);
                }

        /* センス・コードに対する処理 */
            switch(dha_rsd[12])
                {
                case N_TOF:                                            /* ﾉｰ･ﾄﾗｯｸ 0 ﾌｧｲﾝﾄﾞ */
                case P_DF:                                             /* ﾊﾟﾜｰ･ｵﾝ･ﾀﾞｲｱｸﾞﾉｽﾃｨｸ･ﾌｪｲﾗｰ */
                    packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_NOTRDY);
                    return(1);
                case P_O:                                              /* ﾊﾟﾜｰ･ｵﾌ,ﾘｾｯﾄ,ﾊﾞｽ･ﾃﾞﾊﾞｲｽ･ﾁｪﾝｼﾞ */
                    bios_mslc( );                                      /* モード・セレクト・コマンド処理 */
                    if(scsif!=0)                                       /* エラー発生か? */
                        {
                        packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);
                        return(1);
                        }
                    if(ready_h( )==0)                                  /* レディ状態か?  (再帰) */
                        {
                        return(0);
                        }
                    else
                        {
                        return(1);
                        }
                case N_R:                                              /* ﾉｯﾄ･ﾚﾃﾞｨ */
                    bios_ssuc_l( );                                    /* ｽﾀｰﾄ/ｽﾄｯﾌﾟ･ﾕﾆｯﾄ･ｺﾏﾝﾄﾞ処理 */
                    if(scsif!=0)                                       /* エラー発生か? */
                        {
                        packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);
                        return(1);
                        }
                    if(ready_h( )==0)                                  /* レディ状態か?  (再帰) */
                        {
                        return(0);
                        }
                    else
                        {
                        return(1);
                        }
                default:
                    packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL); /* その他のエラー*/
                    return(1);
                }
            }
        return(0);
    }


    /*********************************************************************************/
    /*                     BIOSパケットコマンド設定処理                                  */
    /*                                                                               */
    /*コマンドパケットをBIOSパケットコマンドに                  入力:packet             */
    /*設定する。                                               出力:bios packet        */
    /*********************************************************************************/

void bios_packet_set( )
    {
        bios_packet.transrw_off = packet->up.rw.transrw_off;       /* 転送アドレスをセット */
        bios_packet.transrw_seg = packet->up.rw.transrw_seg;

        bios_packet.sector_count_l = packet->up.rw.sector_count_l;  /* 転送セクタ数セット */
        bios_packet.sector_count_m = packet->up.rw.sector_count_m;
        *((unsigned int *)(&bios_packet.sector_count_l))  *= bscnt;

        bios_packet.start_sector_l  = packet->up.rw.start_sector_l;  /* 開始セクタ番号セット */
        bios_packet.start_sector_hl = packet->up.rw.start_sector_m;
        bios_packet.start_sector_m  = bios_packet.start_sector_hm = 0;
        *((unsigned long *)(&bios_packet.start_sector_l)) *= bscnt;
        return;
    }


    /*********************************************************************************/
    /*                    リクエスト・センス・コマンド設定処理                             */
    /*                                                                               */
    /*リクエスト・センス・コマンドのポインタをchaに,               入力:prog_seg         */
    /*バイト数をtccに設定する。                                  出力:cha, tcc, tdc,    */
    /*リクエスト・センス・データのポインタをdha_seg,                  dha_seg, dha_off  */
    /* およびdha_offに, バイト数をtdcに設定する。                                      */
    /*********************************************************************************/

void rsc( )
    {
        cha.cha_c = &cha_rsc[0];                /* コマンド先頭アドレスをセット */
        tcc = RSC_BC;                           /* トランスファ・コマンド・カウンタをセット */
        dha_off = &dha_rsd[0];                  /* データ先頭アドレスをセット */
        dha_seg = prog_seg;
        tdc = RSD_BC;                           /* トランスファ・データ・カウンタをセット */
        return;
    }


    /*********************************************************************************/
    /*                      モード・センス・コマンド設定処理                              */
    /*                                                                               */
    /*モード・センス・コマンドのポインタをchaに,                   入力:prog_seg         */
    /*バイト数をtccに設定する。                                  出力:cha, tcc, tdc,    */
    /*モード・センス・データのポインタをdha_seg,                      dha_seg, dha_off  */
    /* およびdha_offに, バイト数をtdcに設定する。                                      */
    /*********************************************************************************/

void msc( )
    {
        cha.cha_c = &cha_msc[0];                /* コマンド先頭アドレスをセット */
        tcc = MSC_BC;                           /* トランスファ・コマンド・カウンタをセット */
        dha_off = &dha_msd[0];                  /* データ先頭アドレスをセット */
        dha_seg = prog_seg;
        tdc = MSD_BC;                           /* トランスファ・データ・カウンタをセット */
        return;
    }
    /*********************************************************************************/
    /*                      モード・セレクト・コマンド設定処理                            */
    /*                                                                               */
    /*モード・セレクト・コマンドのポインタをchaに,                 入力:prog_seg         */
    /*バイト数をtccに設定する。                                  出力:cha, tcc, tdc,    */
    /*モード・セレクト・データのポインタをdha_seg,                    dha_seg, dha_off  */
    /* およびdha_offに, バイト数をtdcに設定する。                                      */
    /*********************************************************************************/

void mslc( )
    {
        cha.cha_c = &cha_mslc[0];               /* コマンド先頭アドレスをセット */
        tcc = MSLC_BC;                          /* トランスファ・コマンド・カウンタをセット */
        dha_off = &dha_msld[0];                 /* データ先頭アドレスをセット */
        dha_seg = prog_seg;
        tdc = MSLD_BC;                          /* トランスファ・データ・カウンタをセット */
        return;
    }


    /*********************************************************************************/
    /*                  スタート/ストップ・ユニット・コマンド設定処理                       */
    /*                                                                               */
    /*スタート/ストップ・ユニット・コマンドのポインタをchaに,       入力:なし             */
    /*バイト数をtccに設定する。                                  出力:cha, tcc          */
    /*********************************************************************************/

void ssuc_l( )
    {
        cha.cha_c = &cha_ssucl[0];              /* コマンド先頭アドレスをセット */
```

```
        tcc = SSUC1_BC;                         /*  トランスファ・コマンド・カウンタをセット  */
        return;
    }


    /***************************************************************************/
    /*                  テスト・ユニット・レディ・コマンド設定処理                  */
    /*                                                                         */
    /*テスト・ユニット・レディ・コマンドのポインタをｃｈａに，        入力：なし     */
    /*バイト数をｔｃｃに設定する。                                    出力：cha, tcc */
    /*                                                                         */
    /*                                                                         */
    /*                                                                         */
    /***************************************************************************/

void turc( )
    {
        cha.cha_c = &cha_turc[0];               /*  コマンド先頭アドレスをセット  */
        tcc = TURC_BC;                          /*  トランスファ・コマンド・カウンタをセット  */
        return;
    }

/*************************************************************************/
    /***************************************************************************/
    /*                  リード・エクステンディド・コマンド設定処理                  */
    /*                                                                         */
    /*リード・エクステンディド・コマンドのポインタをｃｈａに，バイト  入力：bios_packet */
    /*数をｔｃｃに設定する。                                  出力：cha, tcc, tdc,   */
    /*リード・エクステンディド・データのポインタをｄｈａ＿ｓｅｇ，      dha_seg, dha_off */
    /* およびｄｈａ＿ｏｆｆに，バイト数をｔｄｃに設定する。                         */
    /***************************************************************************/
void rec( )
    {
        cha.cha_st = &cha_rec;                          /*  コマンド先頭アドレスをセット  */
        tcc = REC_BC;                               /*  トランスファ・コマンド・コウンタをセット  */
        dha_off = bios_packet.transrw_off;              /*  転送アドレスをセット  */
        dha_seg = bios_packet.transrw_seg;
        cha_rec.trans_sector_m = bios_packet.sector_count_m;  /*  転送セクタ数セット  */
        cha_rec.trans_sector_l = bios_packet.sector_count_l;

        tdc = *((int *)(&bios_packet.sector_count_l)) * 512; /* トランスファ・データ・カウンタをセット */
        cha_rec.start_scn_hl = bios_packet.start_sector_hl;  /*  開始セクタ番号セット  */
        cha_rec.start_scn_l  = bios_packet.start_sector_l;
        cha_rec.start_scn_hm = bios_packet.start_sector_hm;
        cha_rec.start_scn_m  = bios_packet.start_sector_m;

        return;
    }


    /***************************************************************************/
    /*                  ライト・エクステンディド・コマンド設定処理                  */
    /*                                                                         */
    /*ライト・エクステンディド・コマンドのポインタをｃｈａに，バイト  入力：bios_packet */
    /*数をｔｃｃに設定する。                                  出力：cha, tcc, tdc,   */
    /*ライト・エクステンディド・データのポインタをｄｈａ＿ｓｅｇ，      dha_seg, dha_off */
    /* およびｄｈａ＿ｏｆｆに，バイト数をｔｄｃに設定する。                         */
    /***************************************************************************/
void wec( )
    {
        cha.cha_st = &cha_wec;                          /*  コマンド先頭アドレスをセット  */
        tcc = REC_BC;                               /*  トランスファ・コマンド・コウンタをセット  */
        dha_off = bios_packet.transrw_off;              /*  転送アドレスをセット  */
        dha_seg = bios_packet.transrw_seg;
        cha_wec.trans_sector_m = bios_packet.sector_count_m;  /*  転送セクタ数セット  */
        cha_wec.trans_sector_l = bios_packet.sector_count_l;

        tdc = *((int *)(&bios_packet.sector_count_l)) * 512; /* トランスファ・データ・カウンタをセット */
        cha_wec.start_scn_hl = bios_packet.start_sector_hl;  /*  開始セクタ番号セット  */
        cha_wec.start_scn_l  = bios_packet.start_sector_l;
        cha_wec.start_scn_hm = bios_packet.start_sector_hm;
        cha_wec.start_scn_m  = bios_packet.start_sector_m;

        return;
    }


    /***************************************************************************/
    /*                  ライト・アンド・ベリファイ・コマンド設定処理                */
    /*                                                                         */
    /*ライト・アンド・ベリファイ・コマンドのポインタをｃｈａに，      入力：bios_packet */
    /*バイト数をｔｃｃに設定する。                                  出力：cha, tcc, tdc, */
    /*ライト・アンド・ベリファイ・データのポインタをｄｈａ＿ｓｅｇ，      dha_seg, dha_off */
    /* およびｄｈａ＿ｏｆｆに，バイト数をｔｄｃに設定する。                         */
    /***************************************************************************/
void wavc( )
    {
        cha.cha_st = &cha_wavc;                         /*  コマンド先頭アドレスをセット  */
        tcc = REC_BC;                               /*  トランスファ・コマンド・コウンタをセット  */
        dha_off = bios_packet.transrw_off;              /*  転送アドレスをセット  */
        dha_seg = bios_packet.transrw_seg;
        cha_wavc.trans_sector_m = bios_packet.sector_count_m;  /*  転送セクタ数セット  */
        cha_wavc.trans_sector_l = bios_packet.sector_count_l;

        tdc = *((int *)(&bios_packet.sector_count_l)) * 512; /* トランスファ・データ・カウンタをセット */
        cha_wavc.start_scn_hl = bios_packet.start_sector_hl;  /*  開始セクタ番号セット  */
        cha_wavc.start_scn_l  = bios_packet.start_sector_l;
        cha_wavc.start_scn_hm = bios_packet.start_sector_hm;
        cha_wavc.start_scn_m  = bios_packet.start_sector_m;

        return;
    }


/*************************************/
/***  BIOS  BPB  INPUT(READ)        ***/
/*************************************/
int     bpb_input( )
{
        bios_bpb_rec( );                        /*  ＢＰＢリード・パケット・コマンド設定処理  */
        scsi_read( );                           /*  ＢＩＯＳリード・コマンド処理  */
        return(stjudg( ));                      /*  コマンド正常終了か？ */
                                                /*  正常終了：０、  エラー：０以外  */
}


void bios_bpb_rec( )
    {
        bios_packet.transrw_off = &work_buf;    /*  転送アドレスをセット  */
        bios_packet.transrw_seg = prog_seg;

        bios_packet.sector_count_m = 0;         /*  転送セクタ数セット  */
        bios_packet.sector_count_l = 1;

        bios_packet.start_sector_hl = bios_packet.start_sector_l = 0; /*開始セクタ番号セット(先頭＝０) */
        bios_packet.start_sector_hm = bios_packet.start_sector_m = 0;

        return;
    }

/*
*       Set   BPB
*/
void    bpb_set( )
{
char    *sptr;
char    *dptr;
int     i;

        sptr = (char *)&work_buf;
        dptr = (char *)&bpb_h;
        for(i = 0; i < 13; ++i) {
                *(dptr+i) = *(sptr+i+11);
        }
}



    /***************************************************************************/
    /*                  ＢＩＯＳリクエスト・センス・コマンド処理                   */
    /***************************************************************************/

int bios_rsc( )
    {
```

〈リスト 5〉 SCDV.C(つづき)

```
        rsc();                          /* リクエスト・センス・コマンド設定  */
        scsi_bios( );                   /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }


    /**************************************************************************/
    /*                  ＢＩＯＳモード・センス・コマンド処理                       */
    /**************************************************************************/

int bios_msc( )
    {
        msc();                          /* モード・センス・コマンド設定処理        */
        scsi_bios( );                   /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }

    /**************************************************************************/
    /*                  ＢＩＯＳモード・セレクト・コマンド処理                     */
    /**************************************************************************/

int bios_mslc( )
    {
        mslc();                         /* モード・セレクト・コマンド設定処理        */
        scsi_bios( );                   /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }


    /**************************************************************************/
    /*              ＢＩＯＳスタート／ストップ・ユニット・コマンド処理              */
    /**************************************************************************/

int bios_ssuc_1( )
    {
        ssuc_1();                       /* スタート／ストップ・ユニット・コマンド設定処理 */
        scsi_bios( );                   /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }


    /**************************************************************************/
    /*              ＢＩＯＳテスト・ユニット・レディ・コマンド処理                  */
    /**************************************************************************/

int bios_turc( )
    {
        turc();                         /* テスト・ユニット・レディ・コマンド設定処理 */
        scsi_bios( );                   /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }


    /**************************************************************************/
    /*              ＢＩＯＳリード・エクステンディド・コマンド処理
                 */
    /**************************************************************************/
int bios_rec( )
    {
        rec( );                         /* リード・エクステンディド・コマンド設定処理  */
        scsi_bios( );                       /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }

    /**************************************************************************/
    /*              ＢＩＯＳライト・エクステンディド・コマンド処理                  */
    /**************************************************************************/
int bios_wec( )
    {
        wec( );                         /* ライト・エクステンディド・コマンド設定処理  */
        scsi_bios( );                   /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }

    /**************************************************************************/
    /*              ＢＩＯＳライト・アンド・ベリファイ・コマンド処理                */
    /**************************************************************************/
int bios_wavc( )
    {
        wavc( );                        /* ライト・アンド・ベリファイ・コマンド設定処理  */
        scsi_bios( );                       /* μＰＤ７２１１１制御処理  */
        return((int)stte);
    }

/*************************************/
/***  SCSI   READ  (BIOS)         ***/
/*************************************/
int     scsi_read( )
{
        bios_rec( );                    /* リード・エクステンディド・コマンド処理  */
        return(stjudg( ));              /* コマンド正常終了か？ */
                                        /* 正常終了：０、  エラー：０以外  */
}


/*************************************/
/***  SCSI   WRITE (BIOS)         ***/
/*************************************/
int     scsi_write( )
{
        bios_wec( );                    /* ライト・エクステンディド・コマンド処理  */
        return(stjudg( ));              /* コマンド正常終了か？ */
                                        /* 正常終了：０、  エラー：０以外  */
}

/*************************************/
/***  SCSI   WRITE & Verify (BIOS) ***/
/*************************************/
int     scsi_writev( )
{

        bios_wavc( );                   /* ライト・アンド・ベリファイ・コマンド処理  */
        return(stjudg( ));              /* コマンド正常終了か？ */
                                        /* 正常終了：０、  エラー：０以外  */
}
```

第6章

自分で作るSCSIのハード&ソフト

ハードディスク&SCSIはアマチュアでも実現可能

宇野沢成夫/橘家鶴蔵/鈴木 洋

最近パソコンにハードディスクは必需品である，という言葉をよく耳にします．

大多数の方々は，市販されているSCSIあるいはSASIのハードディスク・インターフェースに製品のハードディスクを接続していることでしょう．

それはそれでお金を出しさえすればよいのですが，生産が打ち切られた過去のパソコンや，ハードディスクを接続することを前提としていない機種にハードディスクをつなぐためには自分ですべて作らなければなりません．

世の中に「ハードディスク……」というタイトルがついた本はものすごく多いのですが，どうもみんなMS-DOSの使い方の本ばかりでまったく役に立ちません．

そこで自分でハードディスクのシステムを組み上げようと考えている読者へ少しはお役にたてるかもしれない，というのが今回の話題です．

●ハードディスク上のデータ

MS-DOSのファイル管理ディスク・デバイスからのデータの読み書きは数バイトからなるブロック単位で行われます．この単位をセクタと呼び1セクタは通常物理的に256，512，1024，2048バイトが最小単位になっています．

MS-DOSではこれらセクタをいくつかにまとめたクラスタ単位で読み書きします．

ファイルはこのクラスタの番地のどこが使われているかを管理するためのFAT(File Alocation Table)と呼ばれるテーブルでその場所を管理しています．

このFATの場所などは，あらかじめ決められていて図1のようなレイアウトになっています．

フロッピ・ディスクの場合FATエリアの大きさなどはFATの最初のエントリに書いてあるメディア・ディスクリプタを読み，ドライバ中にあらかじめ用意しておいたBPBによって決定するのですが，MS-DOS V3.1以降のハードディスクの場合は通常論理セクタ0の予約領域へBPBのパラメータを書いておき，それをドライバが利用します(図2)．

FATのひとつのエントリにはクラスタの番地が書かれており，エントリが16ビットの場合には最大64K番地を指定することが可能です．1クラスタが512バイトの場合512×64K=約33Mバイト，1クラスタが2048バイトの場合で約134Mバイトの容量までアクセスすることができます．

しかし，これを超える600Mバイトや1Gバイト・クラスのドライブが発売されるようになりました．このような場合には，1台のドライブをいくつかのMS-DOSでアクセスできる範囲の容量の区画(パーテーション)に分け，それぞれを論理ドライブとして使うしかありません．

1台のディスクをいくつかに分けるのですから，その場所がどこからどこまでなのかを管理するテーブルが必要になってきます．これがパーテーション・テーブルです．

また1台のディスクを複数のOSで共有することなどにも，この方法が用いられます．普通パーテーション・テーブルはSCSIの論理ブロック・アドレスの最初のほうにあります．管理情報としては区画の始まりのブロック・アドレス，区画のブロック数，OSの種類などが書かれています．図3はIBM-PCなどで使われているパーテーション・テーブルです．

しかし，パーテーションについては機種によってサポートされているOSが違うことから統一された仕様というものがありません．

これはハードディスクのように取り外しを前提としていないものではあまり問題にならないのですが，光磁気ディスクのように他の機種へ大量のデータを持っていく場合などに問題になります．この問題は現在解決されていません．

フォーマット

ハードディスクを使うためにはさきほど説明したようにMS-DOSなどのOSで使える構造にする必要があります．また，ドライブ+ハードディスク・コントローラ(HDC)といった構成のシステムではHDCで使えるようにドライブをフォーマットする必要があり

ます．前者を論理フォーマットを行うといい，後者を物理フォーマットを行うといいます．

物理フォーマットではディスクの表面をセクタ単位に分け，HDC で読める状態にするのが主な目的ですが，それ以外にもいくつか目的があります．

まず，通常工場から出荷されたドライブには少なからずバッド・スポット(媒体欠陥)が存在します．この部分へデータを書いた場合，データが正常に読み出せなくなる可能性が非常に高くなります．

そこで，物理フォーマット時にその位置を避けるように交代セクタあるいは交代トラックなどのディフェクト処理(代替処理)を行います．これによって，ドライブをアクセスする側からはあたかも連続して正常なセクタが連続してあるように見せることができます．

また，ディスクのインターリーブを変更するのも，この物理フォーマットによって行います．

ただ，最近のコントローラを内蔵した SCSI インターフェースのディスク・ドライブでは，ほとんどトラック単位のバッファを内蔵しているのでインターリーブの変更は逆にスピードの低下をまねくうえ，ディフェクトに関しては工場出荷時点の検査ですでに交替処理は行われているため物理フォーマットは行う必要がなくなってきています．

さらに，ディスク・ドライブの構造もハードセクタを採用してセクタ・サイズの変更ができない物が増えており，SCSI でもこの物理フォーマットを行うコマンドの種類はパラメータが微妙に違うので現在ではユーザが物理フォーマットを行わない方向へ向かいつつあるようです．

もし物理フォーマットを行うのであれば，ディスクが十分暖まった後に行うようにしてください．

これは物理フォーマットだけに限ったことではないのですが，自動車のエンジンをかけていきなりアクセルを全開にすると故障の原因や寿命が縮まるのと同様にハードディスクにも暖気運転は必要です．

論理フォーマットでは MS-DOS などの OS で使用されるファイル構造の初期化やパーテーション・セクタなどのディスクのシステム・エリアを初期化します．

OS-9 の場合にはファイル構造の初期化に使用されるフォーマット・コマンドは RAM ディスクであろう

〈図 1〉 MS-DOS 論理レイアウト

0, 1, 2 …	予約領域
	FAT
	FAT 予備
	ルート・ディレクトリ
	ファイル領域 ↓

〈図 2〉 論理セクタ 0

0–2	ジャンプ命令	
3–10	メーカ名とバージョン	
11–12	セクタ当たりのバイト数	BPB
13	クラスタ当たりのセクタ	BPB
14–15	予約セクタの数	BPB
16	FAT 領域のセクタ数	BPB
17–18	ルート・ディレクトリのエントリ数	BPB
19–20	総セクタ数	BPB
21	メディア・ディスクリプタ	BPB
22–23	1FAT セクタ数	BPB
24–25	1 トラックあたりのセクタ数	
26–27	ヘッド数	
28–29	隠れたセクタの数	

〈図 3〉 IBM-PC のパーテーション・テーブル

オフセット	先頭からの大きさ(バイト)	内　　容
01BEH	16	Partition # 4
01CEH	16	Partition # 3
01DEH	16	Partition # 2
01EEH	16	Partition # 1
01FEH	2	Signature:0AA55H

注) [01FEH]=055H，[01FFH]=0AAH

(a) テーブルの大きさ

オフセット	先頭からの大きさ(バイト)		内　容
00H	1	Boot indicator	80H：ブートする 00H：ブートしない
01H	1	Beginning head	
02H	1	Beginning sector	
03H	1	Beginning cylinder	
04H	1	System indicator	
05H	1	Ending head	
06H	1	Ending sector	
07H	1	Ending cylinder	
08H	4	Starting sector (relative to beginning of disk)	
0CH	4	Number of sectors in partition	

(b) テーブルの内容

コード	意　　味
00H	UnKnown
01H	MS-DOS，12 ビット FAT
02H	XENIX
04H	MS-DOS，16 ビット FAT
06H	DOS4.0
11H	東芝日本語 DOS

(c) (b) の 04H(システム・インジケータ)の内容

とハードディスクであろうとひとつで済むのですが，MS-DOSではシステム・ディスクに付属してくるフォーマット・コマンドはハードディスクのブロック型デバイス・ドライバを書いた場合使えないので，このコマンドを書かなければなりません．

このMS-DOSの場合，フォーマット・コマンドはFATエリアの初期化，ルート・ディレクトリの作成，論理ブロック0にある予約領域へBPBの内容を書き込むといった作業を行います．

また，もし1台のドライブを複数の論理ドライブに分けて使うような仕様にするのであれば，パーテーション・セクタを作成するプログラムも別に書かなければなりません．

MS-DOSのハードディスク・デバイス・ドライバ

MS-DOSで新しいデバイスを使うときにはデバイス・ドライバをCONFIG.SYSに登録することが必要です．MS-DOSではハードディスクのドライバはブロック型デバイスに分類されるので，これを書けば使えるようになります．

ブロック型のデバイス・ドライバの場合MS-DOS側でファイル構造の管理などを行ってくれるので，要求されるコマンドを逐次処理していけばよいうえ，SCSIのハードディスクの場合フロッピ・ディスクとは違い，要求されるセクタ番号などをトラック/ヘッドと分解して制御する必要がないので，非常に簡単に作ることができます．

さらに，MS-DOSはシングル・タスクのDOSなので，他のプロセスを気にせずにポーリングもできるのでプログラムはかなり手を抜いて作ることも可能です．

MS-DOSのドライバの仕様はMS-DOSのテクニカル・マニュアルに出ているので詳しく説明しませんが，SCSIのハードディスクのドライバとして必要になる部分をいくつか書いてみます．

▶イニシャライズ

ドライバのイニシャライズ・コマンドの部分ではCONFIG.SYSに記述されているパラメータを読み込みます．ドライバの仕様にもよるのですが，接続するドライブのSCSI IDやLUNなどをドライバのパラメータとして与えます．当然この部分でインターフェース・チップの初期化，あるいはワーク・エリアの初期化も行います．

SCSIのドライブではそのSCSI IDに接続されているデバイスが本当にディスクであるかチェックするためにSCSIのインクワイアリ・コマンドを発行してチェックするのもよいでしょう．

よく，この部分でSCSIバスにリセットを送るドライバがありますが，SCSIの場合安易にバス・リセットをすべきでありません．なぜなら，複数の機種がSCSIバス上で接続されている場合，それらの機種すべてにリセットが送られてしまうからです．

もちろん，1台しかつながっていなければバス・リセットを送ってもかまわないのですが，通常SCSIのインターフェース・チップはRSTがアサートされると割り込みがかかるような構造になっているので，そのトラップ・ハンドラを先に仕掛けておかなければなりません．

SCSIのハードディスクの場合，このRSTがアサートされた後，あるいは電源投入直後に発行されたコマンドのステータス・フェーズは必ずユニット・アテンション(06H)になります．

そのためこれをクリアするにダミーのリクエスト・センス・コマンドを発行します．

ハードディスクの場合，この部分で物理セクタのサイズを取得したり，パーテーション・テーブルを読み込むといったことを行ってもよいのですが，メディアの取り外しのできる光磁気ディスクなどのリムーバブル・ディスクなどはこのルーチンはドライバがインストールされたときに一度しか呼ばれないのでこの時点ではできません．ですから最初のアクセスかどうかのフラグを設けておきBUILD BPBルーチンなどで行うべきでしょう．

▶メディア・チェック

CCS準拠のドライブの場合，リムーバブルなメディアであるかどうかはインクワイアリ・コマンドで知ることができます．

このルーチンはリード/ライトなどの前に呼ばれるので，リクエスト・センス・コマンドなど無難なコマンドを発行することで，もしメディアが交換された場合にはステータス・フェーズでユニット・アテンション(06H)が報告され，以前入っていたディスクと違うものであることがわかります．

▶BUILD BPB

もし簡単に済ませるのならドライバ内にテーブルを用意しておいてもよいのですが，論理フォーマットのところでも説明したようにMS-DOSの論理セクタ0の部分に書いてあるのであればそれを読んできて返すことで実現できます．

また，このルーチンはドライブ番号を与えられて初めてアクセスするときには必ず呼ばれるので，ドライブをいくつかの区画に分けて使う場合パーテーション・テーブルから実際の物理セクタ番号を計算するためのオフセットなどをワーク・エリアへセットする操作などもここで行います．

▶オープン/クローズ

MS-DOS V3.1以降でサポートされたファンクショ

ンでデバイス・ヘッダのオープン/クローズ/リムーバブル・メディアのビットを立てた場合にアプリケーションがファイルをオープンあるいはクローズのシステム・コールを実行したときに呼ばれます．

普通，メディアの取り外しができないハードディスクなら無視してもかまわないのですが，光磁気ディスクなどメディアが交換できるドライブではこのファンクションを使ってメディアの取り外しができないようにするために使います．

具体的にはオープン・ファンクションが呼ばれるたびに値が増えるカウンタをワーク・エリアに用意しておき，オープン・ファンクションが呼ばれたときにリムーバブル・メディアでかつオープンの回数がゼロのときドアをロックしてメディアをイジェクトできないようにします．

クローズ・ファンクションではこの逆にカウンタ値をまず減らし，ゼロならドア・ロックを解除します．

書き込みオープンした状態でメディアを取り出した場合普通はファイルがクラッシュするので，もしリムーバブルなディスクを使用するのであれば積極的に利用したいファンクションです．

▶リムーバブル・メディア

これも MS-DOS V3.1 以降にサポートされたファンクションです．オープン/クローズ/リムーバブル・メディアのビットが立てられているときに呼ばれます．メディアがリムーバブルかどうかはすでにイニシャライズ・ルーチンで SCSI のインクワイアリ・コマンドで判明しているので単にその結果を返すだけです．

▶リード/ライト/ベリファイ付きライト

実際の読み書きを行うルーチンです．このルーチンでいかに速く転送を行うかによってディスクのスピードが決まってしまいます．

マルチ・タスクの OS などでは本来 DMA 転送を行うべきなのでしょうが，シングル・タスクの MS-DOS ではあまりそのメリットがあるとはいえません．パラメータとして渡されるのは論理セクタ・アドレスですからこれをドライブの物理セクタ・アドレスに変換し，パーテーションを切ってある場合にはそのオフセットを加えます．

また，データの信頼性については，ハードディスク

〈図 4〉インクワイアリ・コマンドとインクワイアリ・データ

バイト＼ビット	7	6	5	4	3	2	1	0
0	オペ・コード (12H)							
1	論理ユニット番号			保留				
2	保留							
3	保留							
4	割り当て長							
5	ベンダ・ユニーク		保留				フラグ	リンク

(1) 保留の部分は，通常 0
(2) フラグとリンクは，コマンドを連結して実行するときに使用
単独で実行するときは，フラグ＝0，リンク＝0

(a) インクワイアリ・コマンド

バイト＼ビット	7	6	5	4	3	2	1	0
0	周辺装置機種							
1	RMB	機種限定						
2	ISO バージョン		ECMA バージョン			ANSI バージョン		
3	保留							
4	追加長 (n)							
5～$n+4$								

ベンダ・ユニーク・パラメータ
(追加長に示されるバイト数のデータ)

・周辺機器機種

00H　直接アクセス装置(例：磁気ディスク)
01H　順次アクセス装置(例：磁気テープ)
02H　プリンタ
03H　プロセッサ
04H　単一書き込み多重読み取り装置(例：1 回書き込みの光ディスク)
05H　読み取り専用直接アクセス装置(例：光ディスク)
06H～7EH　保留
7FH　論理ユニットが存在しない．
80H～FFH　ベンダ・ユニーク

(b) インクワイアリ・データ

〈図 5〉DTC-310DB の SASI コマンド

		SC601	DTC 系	XEBEC 系
コマンド		C2H	C2H	0CH
パラメータ	0	デバイス・モード	01H	シリンダ数(MSB)
	1	00H	ステップ周期	シリンダ数(LSB)
	2	00H	ステップ・モード	ヘッド数
	3	00H	ヘッド数	RWC(MSB)
	4	00H	シリンダ数(MSB)	RWC(LSB)
	5	ヘッド数	シリンダ数(LSB)	WPC(MSB)
	6	シリンダ数(MSB)	RWC(LSB)	WPC(LSB)
	7	シリンダ数(LSB)	RWC(MSB)	ECCバースト・ビット
	8	RWC(MSB)	00H	—
	9	RWC(LSB)	00H	—
	10	WPC(MSB)	—	—
	12	WPC(LSB)	—	—

RWC：Reduce Write Current Cylinder
WPC：Write Precompensation Cylinder

SC601 デバイス・モード
ビット 2："0" DK-511 シリーズ，"1"その他
ビット 0："0" 256 バイト/s，"1" 512 バイト/s
他のビットはすべて "0"

の場合ECCによるエラー・コレクションが行われるので通常転送されてくるデータが化けてくることはまずあり得ません．もし，エラーが起こりECCによって修正された場合はステータス・フェーズでリカバリ・エラー(01H)になります．この場合完全に読み出すことができなくなる前に代替処理を行うべきでしょう．ECCでも修正できなかった場合には通常はデータが送られてきません．

ベリファイについてはSCSIのコマンドのベリファイを用いて行います．ドライブによっては書き込んだデータがトラック・バッファに蓄えてあるのですぐに同じアドレスから読み込んでベリファイを行う方法ではデータをメディアから読み込むのではなくメモリにあるデータを転送してくる場合もあるので，あまり意味がありません．

ドライバを書く上で必ず使うSCSIのコマンドは，いうまでもなくリード(08H)とライト(0AH)ですが，それ以外にもドライバの機能として必要なものがいくつかあります．

まず，OSにもよるのですが，ドライバの初期化ルーチンで接続されているドライブの物理ブロック・サイズなどある程度の仕様を知る必要があります．SCSIのドライブではリード・キャパシティ(25H)によって物理ブロック・サイズと最大ブロック・アドレスを知ることができます．

また，光磁気ディスクなどのリムーバブル・メディアを使用する可能性がある場合やSCSIバスに接続された，使用しようとするデバイスが本当にハードディスクなどのダイレクト・デバイスであるか，といったチェックにはインクワイアリ(12H)というコマンドを使用します(図4)．

リムーバブル・ディスクの場合は，通常メディアの取り出しを禁止するためのコマンドPREVENT/ALLOW MEDIUM REMOVAL(1EH)がサポートされていているので，さきのオープン/クローズなどで利用します．

一般的にSASIのドライブではこれらのコマンドはサポートされていないため，前もって物理ブロックを使用できるサイズにしておくなどの対策が必要です．

その他SASIの場合，ドライブを最初にアクセスするときにドライブの仕様を知らせる必要がある場合もあります．

具体的にはアサイン・ディスク・パラメータという名前のコマンドを発行します．

SCSIの場合，通常はドライブの記録面の数シリンダ分をシステム領域としてそのメディアに仕様やディフェクトなどの情報を記録しておくようになっていますが，それより前に製品として登場したSASIではそのような処理は行われず，ボード上のジャンパの設定やファームウェアのROMのテーブルなどによりデフォルトのシリンダ数やヘッド数などを持っています．

OEM向けのコントローラ・ボードの場合，ドライブの仕様などは決定されているのでそのデフォルト値でなにも問題なく使えますが，組み合わせるドライブが別のコントローラ・ボードの場合にはそれとは違った仕様のドライブを接続する可能性のほうがあるくらいです．このコマンドはコントローラ・ボードによりコマンド・コードやパラメータがまちまちです．

代表的なSASIのハードディスク・コントローラ・ボードのコマンドは図5のようになっています．

〈図6〉センス・データ

バイト＼ビット	7	6	5	4	3	2	1	0
0	AdValid	誤り区分			誤りコード			
1	製造者規定			論理ブロック・アドレス(MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス(LSB)							

(a) 非拡張センス・データ形式

バイト＼ビット	7	6	5	4	3	2	1	0
0	AdValid	誤り区分			誤りコード			
1	セグメント番号							
2	ファイル・マーク	EOM	ILI	保留	センス・キー			
3	情報バイト(MSB)							
4	情報バイト							
5	情報バイト							
6	情報バイト(LSB)							
7	追加センス長							
8−n+7	追加センス・バイト							

(b) 拡張センス・データ形式

ステータス・フェーズで00H(ノー・センス)以外の場合はなんらかの問題があったときです．このときドライバではリクエスト・センス(03H)コマンドでこれをクリアすると同時に原因を探り，適切なエラーを返すのですが，このコマンドによって返されるデータは**図6**のように2種類あります．

現在発売されている製品ではSCSIのドライブではエラー・クラス7の拡張センス・データ形式，SASIのドライブでは4バイトの非拡張センス・データ形式を返すようです．この非拡張センス・データ形式はドライブ(コントローラ)によってエラー・コードがまちまちなので，ドライバはエラー時の処理をその原因を作ったコマンドによって判断する方法をとるほうが無難です．

SCSI/SASIのコントローラ・ボード

最近のドライブはSCSIのコントローラ内蔵のものが主流になってきているのでもはや死語になった感があるのですが，ディスク・ドライブ＋ハードディスク・コントローラの組み合せのディスク・システムは最近リース期限が切れて秋葉原などでジャンクとしてかなり出まわるようになってきました．

当時は20Mバイトあたりが主流だったので現在の肥大化したアプリケーションではとても容量が足りないうえにスピードも現在のものに比べかなり遅く感じます．

ところが，値段はフロッピ・ディスクでその容量分を買うよりも安い場合があるので(ジャンクですから当然リスクを考えなければなりませんが)バックアップ用として利用するのにはよいかもしれません．

そこで，簡単ですが秋葉原などに出まわる可能性のあるハードディスク・コントローラをいくつか紹介してみます．

▶ SC601-1/SC601-2/SC601-3(**写真1**)

日立のSASI-ST506のMFMのコントローラです．4台までのドライブを接続することができトラック・バッファを内蔵しているので，比較的高速で転送が行えます．

通常このコントローラは同社の5インチのドライブであるDK511シリーズとの組み合わせで使われる設定が施されており，他のドライブで使用する場合はジャンパの設定を変更する必要があります．

変わったところではSC601-1が同社のST506インターフェースのフロッピ・ディスク・ドライブFDD-411(8インチ・フォーマット容量6.15Mバイト)およびSC601-2およびSC601-3がFDD-451(5インチ・フォーマット容量4.15Mバイト)に接続できます．

▶ DS300/DS500

NECのSASI-ST506のMFMのコントローラです．同社のパソコンPC9801シリーズの普及により国内ではこのコントローラの工場出荷時の設定コマンド体系がかなり一般的なものになっています．

DS300は3.5インチ・サイズで8ヘッドまで，DS500が5インチ・サイズで16ヘッドまでのドライブが制御可能です．

これらのコントローラはディップ・スイッチの設定によってDモード/Xモード/Sモードの3種類のコマンド体系が選択できるようになっています．

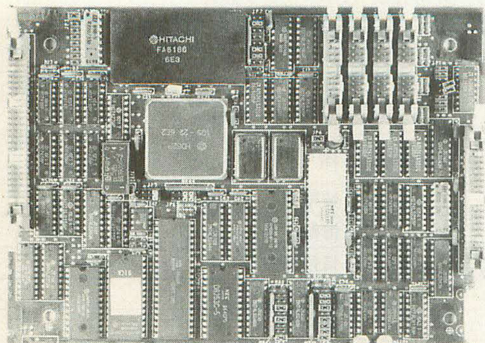
それぞれアメリカの有名なコントローラ・メーカの頭文字をとったネーミングのようで，だいたいどのメーカのコマンドとコンパチか想像がつくと思います．

工場出荷時はDモードに設定されており，サード・パーティのPC9801シリーズ用SASIのハードディスクではこれに習ったコマンド体系になっているようです．

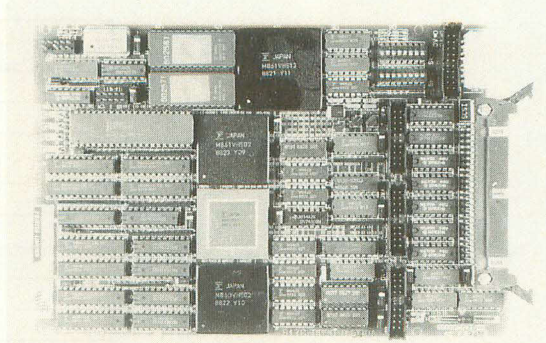
▶ M1055B

富士通のSCSI-ST506のMFMのコントローラです．初期のFM-Rシリーズのハードディスクで使われていたコントローラで，コマンド体系はCCSの規格をほとんど満たしています．

このコントローラはNCL社よりOEM供給されていたものでNCL社ではNDC-3016という型番で呼ば



〈写真1〉 SC601-1



〈写真2〉 M1054A

〈表 1〉
SPC の種類

	MB87030	MB87031	MB87033	MB89351	MB89352
ドライバ/レシーバ	外付け	外付け	内蔵	外付け	内蔵
同期転送	○	○	○	×	×
転送カウンタ	24 ビット	24 ビット	28 ビット	24 ビット	24 ビット
パッケージ	88 ピン PGA	100 ピン QFP	80 ピン QFP	64 ピン QFP 64 ピン SH-DIP	48 ピン QFP 48 ピン DIP

れています.

▶ M1054A(**写真 2**)

富士通の SCSI-ESDI のコントローラです. 一応 SCSI の規格を満たしてはいるものの, CCS のコマンド体系ではありません.

接続可能ドライブは同社 M2244CA/M2245CA/M2246CA の 3 機種に限られているようです.

▶ ACB4000/ACB4070

米アダプテック社の SCSI-ST506 のコントローラです. 記録方式は ACB4000 が MFM, ACB4070 が RLL です.

このコントローラはある程度 CCS コマンド体系をサポートしているものの, フォーマットやディフェクト処理などのコマンドについては CCS と違っているので注意が必要です.

アメリカではマッキントッシュなどのハードディスクでかなり使われているらしく, かなりメジャーなコントローラのようです.

▶ ACB4520

米アダプテック社の SCSI-ESDI のコントローラで先の ACB4070 などより後から設計されたものらしく, コマンド体系は CCS に完全に対応しています.

▶ M1052A/M1052B(**写真 3**)

富士通の MDC-SA4000 のコントローラです. SCSI インターフェースではないので注意が必要です.

このコントローラは富士通の FM11 や FM16β などのハードディスク用として使われていたもので, M1052A が 20M バイトまでのステッピング・モータを使用している SA4000 バスのドライブを, M1052B はそれらを含み 67M バイトまでのものをコントロールすることができます.

SA4000 のドライブというのは富士通のドライブの型番でいうと M2235B, M2243B といったラスト・レターが B のもので, コネクタが 50 ピンのカード・エッジになっているものです. なお, MDC のコネクタも同じ 50 ピンなので, SCSI と勘違いして買ってしまいそうですが, これらを SCSI のインターフェースに接続しても何があろうと動きません.

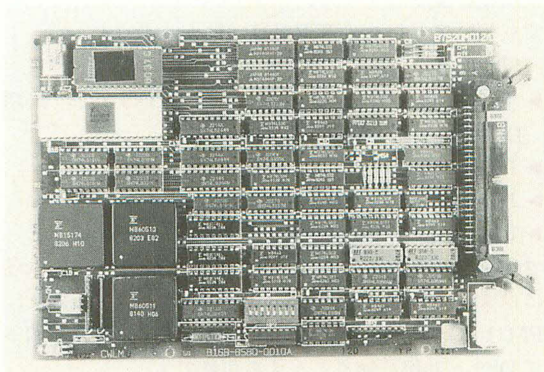
SCSI インターフェース

SCSI のコントローラ・チップはいくつか発売されていますが秋葉原などで比較的入手しやすいチップは富士通の MB89352 と NCR 社の NCR5380 です.

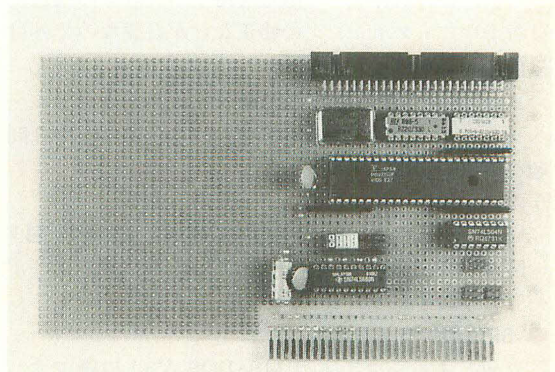
NCR5380 はかなり前から発売されていただけあって, いろいろな記事に紹介されているので説明の必要はないでしょう. NCR5380 の場合 SCSI のフェーズをソフトウェアで実行しなければならず, SCSI のプロトコルを理解するにはよいのですが, パフォーマンスは若干落ちます.

MB89352 は MB87030 という同社の SCSI コントローラ・チップから, 同期転送機能を外し SCSI の不平衡ドライバを内蔵したもので, 最近では J3100 シリーズや X68000 シリーズの SCSI インターフェース・カードにも搭載されるようになり, かなりメジャーなチップになってきています.

特徴はデータ転送時にデータ・バッファに 8 バイトの FIFO バッファと 24 ビット・カウンタを持っているため転送を高速に行えることです. なお, 富士通ではこの SCSI コントローラを SPC(SCSI PROTOCOL

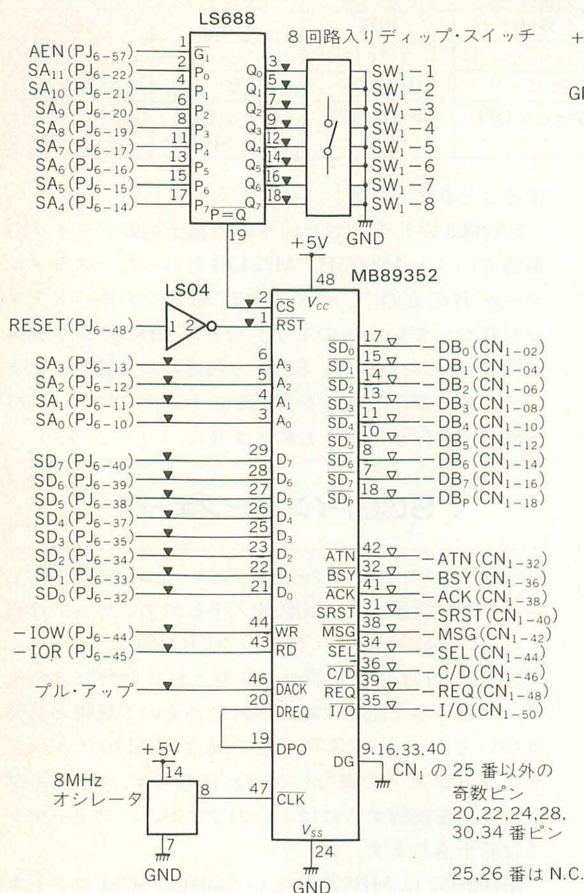


〈写真 3〉 M1052A



〈写真 4〉 SPC ボード

〈図7〉 J3100用SPCボード



+5V(PJ6-02)
(PJ6-52)

GND(PJ6-01)
(PJ6-09)
(PJ6-18)
(PJ6-27)
(PJ6-43)
(PJ6-51)
(PJ6-60)



▶信号線とコネクタの対応を書いておきます．

AEN(A_{11})	SA_{11}(A_{20})	SA_{10}(A_{21})	SA_9(A_{22})	SA_8(A_{23})
SA_7(A_{24})	SA_6(A_{25})	SA_5(A_{26})	SA_4(A_{27})	SA_3(A_{28})
SA_2(A_{29})	SA_1(A_{30})	SA_0(A_{31})	−IOW(B_{13})	−IOR(B_{14})
SD_7(A_2)	SD_6(A_3)	SD_5(A_4)	SD_4(A_5)	SD_3(A_6)
SD_2(A_7)	SD_1(A_8)	SD_0(A_9)	RESET(B_2)	
+5V(B_3,B_{29})	GND(B_1,B_{10},B_{31})			

ディップ・スイッチの1～8が，I/OアドレスのA_{11}～A_4に相当しています．Onでそのアドレスが0Offで1になります．
ですから，例えばアドレスを0300Hに設定するには，1番から順番にOn On Off Off On On On Onとディップ・スイッチを設定します．

▶回路がちゃんとつながっているのに動かない，という場合，半分くらいは電源，グラウンドの配線が弱いためです．まず最初に部品の配置を決めて，その後グラウンド，電源を強力に配線して，その後信号線を配線するようにしましょう．
私は通常は「はんだ吸い取り線」を使ってグラウンド，電源を配線しています．また，パスコンではICのすぐ脇に実装する事も忘れずに！今回の場合，終端抵抗にもパスコンを入れた方が良いと思います．

CONTROLLER)と呼んでおり，シリーズにはこのMB89352の他に**表1**のような種類があります．

● 実際の回路

MB89352の場合，いわゆる80バスの信号インターフェースによって制御されるので，86のバスを持つパソコンではチップ・セレクト信号を作るくらいで非常に簡単にSCSIインターフェース・カードができてしまいます．

ただ，それ以外のバス信号を持つパソコンの場合には若干バス・タイミングの修正などが必要なので外付け部品が必要になります．

また，DMA転送を行う場合リクエストをレベル・センスによって転送を行うDMAコントローラでは問題ないのですが，エッジ・トリガが必要なものでは若干の工夫が必要になります．

いくつかのパソコンで接続する場合の回路図を**図7**～**図9**に示します．また，製作したIBM用SPCボードを**写真4**に示します．

● SPCのドライバ

対象とするシステムはMS-DOS V3.1以降です．機種は問いません．**リスト1～リスト7**に全プログラム(ドライバ，フォーマット・プログラム)を示します．MAKEFILEを実行することで.SYSと.EXEができます．

ただし，FATの大きさを4085以下にすればV2.1でも動作します．

8086/186コードで書かれています．使用したアセンブラはMASM4.0/5.1です．

DMAを使用していないので，マシンのハードにほとんど影響されません．MS-DOSマシンならばどれでもI/Oアドレスの変更だけで動くはずです．

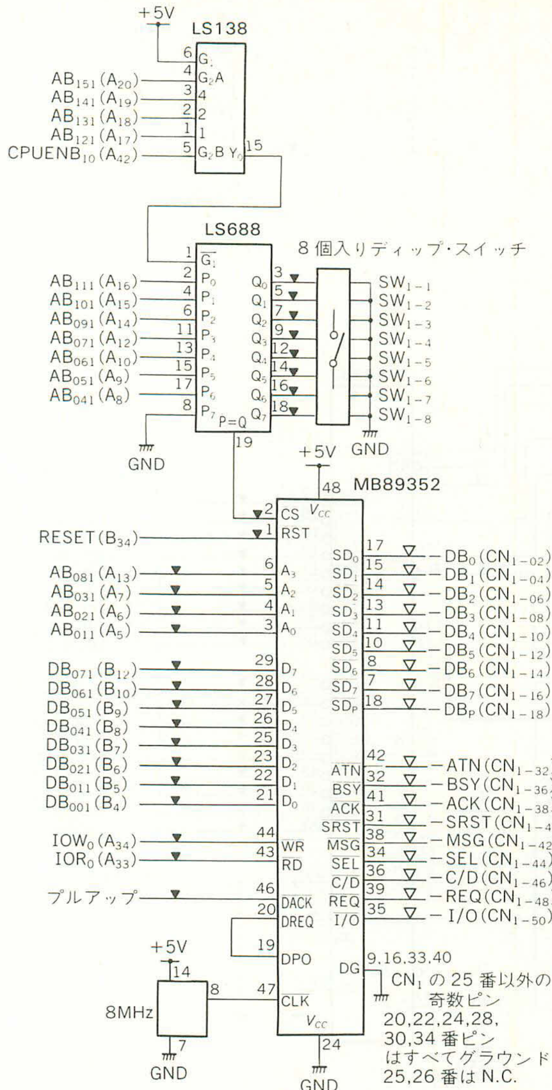
現在のところ，

▶ J3100SGT/GX/SL/SS(英語DOS3.3/日本語DOS3.1)
▶ PC9801VX41(MS-DOS3.3)
▶ IBM-PC/XT，JX，5550(DOS3.3)
▶ FM11AD2＋(MS-DOS2.1/3.1)

で動作確認されています．どれも，I/Oアドレス以外にはまったく手を加えずに動作しています．また，FM11では現在同一ドライブをパーテーションを切ってOS9と共用で使うことすら可能です．

対象になるドライブはANSIの標準に準拠した

〈図8〉PC9801用SPCボード





▶ディップ・スイッチの1〜8がI/OアドレスのA_{11}〜A_4に相当しています.
OnでそのアドレスがO.Offで1になります.A_{15}〜A_{12}は0に固定です. ですから, 例えばアドレスを02E0Hに設定するには, 1番から順番にOn OnOff On Off Off Off On とディップ・スイッチを設定します.

▶PC9801は他の機種と違って8ビット・バスで奇数番地I/Oができません. このため, アドレスは2番地おき, 上位8バイトと下位8バイトは100H番地離してロケーションしています(SPCのA_3に, バスのA_8が入っている). また, I/Oアドレスはフル・デコードしています.
それ以外はJ3100用と同じです. 製作上の注意については, J3100用回路図を参考にして下さい.

SCSIドライブ(CCSが使えるドライブ), SONYおよびそのOEMのMOディスクです.

具体的にはNEC PC9801-55カード(あるいは同等品)に接続できるドライブ, および富士通FM-R/TOWNSシリーズに接続できるドライブなどであればまったく問題なく使用できます.

PC9801-27カードに接続できるドライブについては, イニシャライズ・ルーチンとステータス・チェック・ルーチンを書き替えれば動作可能ですが, このままでは動かないと思います. 特にTDC384などはROMがいろいろあるらしいので…….

Mac用については仕様がいろいろあるそうなので, すべてが変更なしに接続できるという保証はできませんが, 動かなくても一部パッチを当てれば動くと思います(ただし未確認). MOはMac用として売られているのはSONY製が多いようなので多分動くと思います.

● ドライバの使用方法

SPCカードのハードウェアができあがったらカードのI/Oアドレスを空いているアドレスに設定し, SFORM.EXEを実行してください.

IBM-PC互換機では0300H, 98互換機では02E0Hあたりがとりあえず適当だと思います. このSPC.SYSでは0300Hになっています.

SFORMはMS-Cで書いてあり, SCSIのハンドシェイクもすべてCでやっているので, 実行時にSPC.SYSがある必要はありません.

なお, SFORMはPC9801の場合はI/Oアドレスを変更して再コンパイルが必要です(8ビット・バスでの奇数番地のI/Oアクセスができないため). その他の機種の場合はこのままで実行可能なはずです.

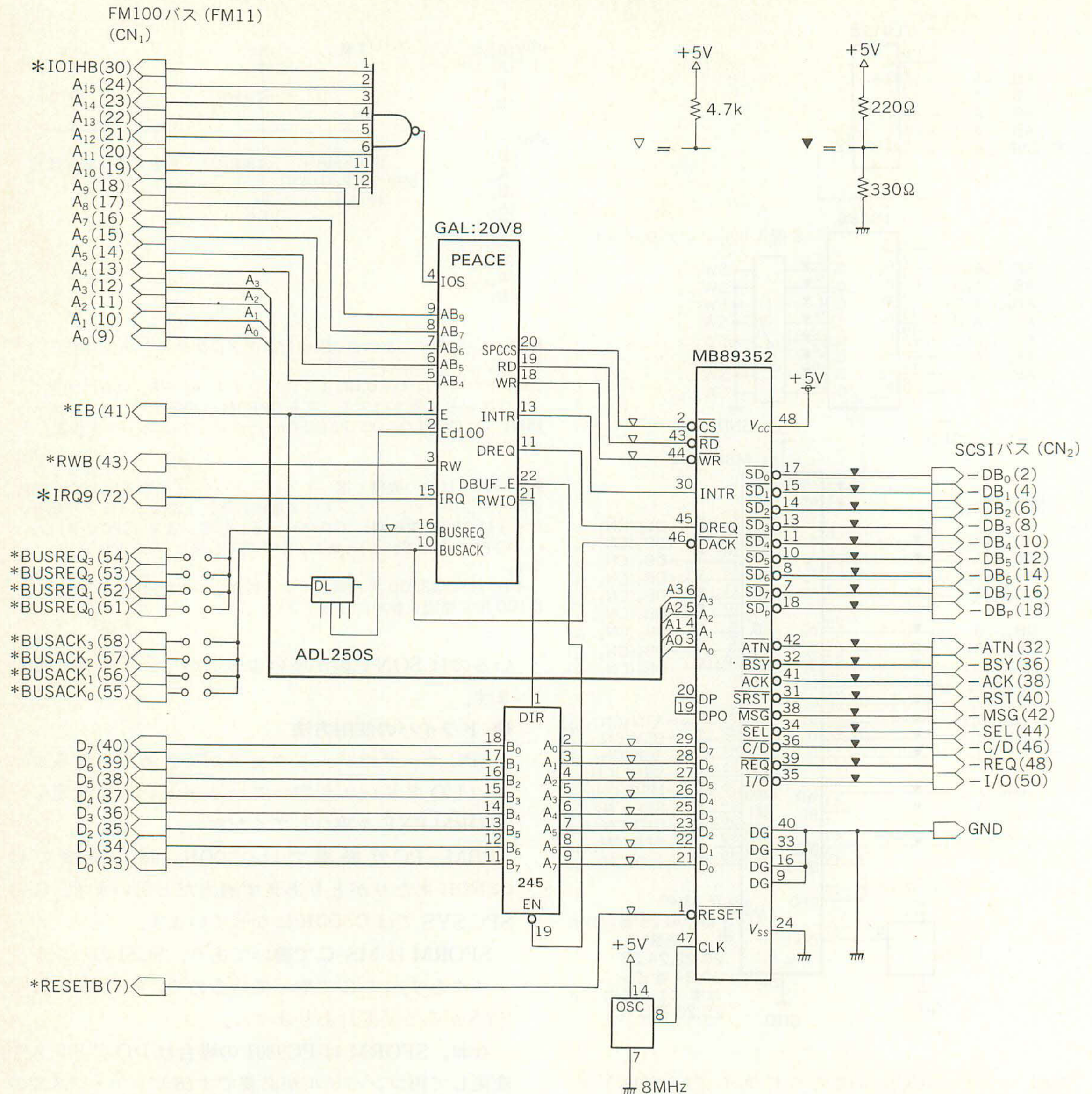
これが正常終了したら, ハードウェアは多分正常なので, つぎにSPC.INCの中のI/Oアドレスを各自のマシンのアドレスに書き替えて再アセンブルしてください.

あとはCONFIG.SYSにSPC86.SYSまたはSPC186.SYSを組み込めば終わりです.

CPUが8086・80C86の場合はSPC86.SYSを, V20/30/186/286/386の場合はSPC186.SYSを使ってください. SPC86.SYSは当然186や286でも動きますが, スピードが10%程度は落ちます.

なお, 起動時に論理ブロック0を読み込んでパーティション情報などをセットしているので, 起動時には

〈図 9-a〉 FM11 用 SPC ボード



〈図 9-b〉 FM11 用 SPC ボードの GAL

```
                                                              Page 1
ABEL(tm) Version 2.10a  -  Document Generator         30-Aug-89 01:07 PM
SPC Card Decorder & Timming Generator
1989/06/08  Desined by TheEND, Named by AZUKI
Equations for Module _PEACE

Device PEACE


- Reduced Equations:

    ~SPC_CS = !(AB4 & AB5 & !AB6 & AB7 & !AB9 & !~IOS);

    ~DBUF_E = !(!~BUSACK & !~E
                # !~BUSACK & !~Ed100
                # AB4 & AB5 & !AB6 & AB7 & !AB9 & !~E & !~IOS
                # AB4 & AB5 & !AB6 & AB7 & !AB9 & !~Ed100 & !~IOS);

RWIO = (!~BUSACK & !RW # ~BUSACK & RW);

~RD = !(!~BUSACK & !~E & !~Ed100 & !RW # ~BUSACK & !~E & !~Ed100 & RW);

~WR = !(!~BUSACK & !~E & !~Ed100 & RW # ~BUSACK & !~E & !~Ed100 & !RW);

enable ~BUSREQ = (~BUSACK & DREQ);

~BUSREQ = !(1);

enable ~IRQ = (INTR);

~IRQ = !(1);
```

必ずHDDがレディ状態になっている必要があります．

MOの場合にはデバイス番号を取得する必要上，ブート時にはMO媒体が入っている状態にしておいてください．入れておかないとイニシャライズ・エラーになります．

CONFIG.SYSにパラメータが入れられるのでIDとLUNが設定可能です．

I/Oアドレスは固定になっています．ですからこれらを変更したい場合にはソースを直して再アセンブルする必要があります．

アドレスはSPC.INCで定義しています．現在のI/Oアドレスは0300H番地から，デフォルトのターゲットIDは0，LUNは0です．

このドライバ単体ではマルチ・ターゲット(複数台のHDD)はサポートしていませんが，CONFIG.SYSでSCSI IDかLUNを変更してドライバを複数回登録すれば複数台接続も可能です．若干メモリの無駄になりますが，1ドライブあたり1.5Kバイト程度なので我慢できると思います．

CONFIG.SYSの例：

```
DEVICE=SPC186.SYS /I0 /L0
DEVICE=SPC186.SYS /I0 /L1
```

上の例では，ID0，LUN0とID0，LUN1の2台のHDDが使えるようになります．

ターゲットIDというのはSCSIコントローラ(ドライブ)のSCSIバス上でのIDです．7はパソコン側が使うので，0から6までです．

LUNというのは，SCSIコントローラとハードディスク・ドライブ間での機番です．ST506インターフェースのHDDの場合，ドライブの機番が1から4まで選べますが，これが1のときにLUN=0です．SCSI内蔵型HDDの場合は，必ず0になります．

パーテーションは12まで切れます．それぞれ最大32Mバイトまでです．ドライバ・イニシャライズのときにドライブの論理セクタ0(DDセクタ)を読み込んで，それから判断するので，ブートの際に必ずデバイスがレディになっている必要があります．

MS-DOSには，「標準で組み込まれているデバイスよりも大きなセクタ・サイズのブロック・デバイスは追加できない」という闇の掟があります．このため，IBM-PCなどでは512バイト/セクタ以上にできないので，32Mバイト以上のパーテーションが切れません．日本語DOSしか絶対に使用しないというのならば1024バイト/セクタもできるので，64Mバイトのパーテーションも可能です．

DDセクタの構成はつぎの通りです(論理ブロック0)．

2バイト：パーテーションの個数
4バイト＊12：それぞれのパーテーションの開始する論理ブロック・アドレス
14バイト：未使用
16バイト＊12：パーテーションごとのBPB情報
合計256バイト

DMA転送はサポートしていませんが，CPUの性能さえ十分であれば通常のHDDよりはるかに高速でアクセスできます．

J3100GXの286-12MHz時は実測でデータ・フェーズの転送レートが560Kバイト/s出ており，PC9801のDMA転送時(実測300Kバイト/s)よりも約1.8倍高速です．つまりは，286-8MHz以上ならば8237を使ったDMA転送には絶対負けません．V30-8MHzで同等のようです．

以下にベンチ・マーク・テストの結果を示します．

内容：3788800バイトのファイルの同一HDD内でのコピー

J3100GX(80286-12MHz) +FMHD-A11	19秒
J3100GX(80286-12MHz) +M1055B+D3126(インターリーブ2)	44秒
PC9801VX41(80286-10MHz) +FMHD-A11	18秒

参考までに，純正HDD&ドライバでは(すべてDMA使用)以下の通りの結果でした．

J3100GX(80286-12MHz) +内蔵HDD(コナー40Mバイト)	34秒
PC9801VX41(80286-10MHz) +内蔵HDD(D3126)	79秒
PC9801VX41(V30-8MHz) +内蔵HDD(D3126)	82秒
FMR70HX2(80386-20MHz) +FMHD-A11	20秒

D3126というのは，NECの3.5インチHDD(20Mバイト)です．FMHD-A11というのは，富士通製のFM-Rシリーズ用外付けHDD(130Mバイト)です．M1055BというのはFM-Rシリーズの内蔵HDDやFMHD-411/611などに使われているSCSIコントローラです．

トラック・バッファがないコントローラの場合，インターリーブ値によって転送速度が大きく変化します．また，256バイト/セクタの場合512バイト/セクタのときに比べて若干データ転送レートが落ちます．つまり，512でインターリーブ2では駄目でも256だと間に合う場合もあるわけです．

このあたりの事情は，上の結果を見てもわかる(12MHzのJ3100GXよりも，10MHzの98の方が速い，しかし内蔵HDDはJ3100の方が2倍も速い)とおり，千差万別なので実機でテストしてみるしかありません．

このドライバはMOに対応していますが，パーテーションの切り方を変えた媒体を入れ替えたときに問題が生じます．これはMS-DOSのシステム側の手抜きが原因なのですが，逃げる方法としてはBPBのメディア・ディスクリプタ・バイトをパーテーションが違うもの同士でかみ合わないよう変更する以外ありません．

パーティションの切り方を変えた媒体を入れ替えたら，リセットするのが一番簡単ですが，もし「どうしても！」という方がいらっしゃいましたら筆者までご相談ください．

なお，同じ切り方の媒体でしたら，いくら入れ替えても大丈夫です(書き込み中などを除く)．

● SPCのフォーマット・プログラム

SPCカード用物理フォーマット/パーテーション・ユーティリティです．

MS-Cで書いてあります．I/Oの操作もすべてCでやっているので，使用する際にSPC.SYSが登録されている必要はありません．

I/Oアドレス，SCSI ID，LUNは最初に手で入力するようになっているので，PC9801以外ではそれらを変更して再コンパイルする必要はありません．

媒体検査でエラーになった個所は自動的にディフェクトとして登録され，交替処理が行われます．

ジャンク屋などで買ってきたドライブは初回の物理フォーマット後に媒体検査を多数回(できれば10回位以上)やっておくと，比較的安心して使えます(特に，ディフェクト・リストが付いていなかった場合)．

M1055B+D3126で，初めてフォーマットを行う場合の操作例を以下に示します．ここでは，

▶ SPCカードのI/Oアドレスは0FDF0番地

▶ セクタ・サイズは256バイト/セクタ

▶ SCSI ID，LUNは0，ディフェ

〈図10〉パラレル・インターフェース・ブロック図

〈図11〉Easy Hard用ケーブル

《DynaBook》プリンタ・ポート		《ハードディスク》SCSIバス
－STROBE(1)	—	(44)－ACK
D_0(2)	—	(26)－D_0
D_1(3)	—	(27)－D_1
D_2(4)	—	(28)－D_2
D_3(5)	—	(29)－D_3
D_4(6)	—	(30)－D_4
D_5(7)	—	(31)－D_5
D_6(8)	—	(32)－D_6
D_7(9)	—	(33)－D_7
－ACK(10)	—	(49)－REQ
BUSY(11)	—	(46)－MSG
PE(12)	—	(43)－BSY
SELCT(13)	—	(48)－C/D
－AUTFD(14)	—	(47)－SEL
－ERROR(15)	—	(50)－I/O
－IPRT(16)	—	(41)－ATN
－SLIN(17)	—	(45)－RST
GND(18～25)	—	(1～12,14～25)GND
DSUB25J		AMPHENOL50J(シールド)

(a) HDD側のコネクタがアンフェノール・タイプの場合

《DynaBook》プリンタ・ポート		《ハードディスク》SCSIバス
－STROBE(1)	—	(38)－ACK
D_0(2)	—	(02)－D_0
D_1(3)	—	(04)－D_1
D_2(4)	—	(06)－D_2
D_3(5)	—	(08)－D_3
D_4(6)	—	(10)－D_4
D_5(7)	—	(12)－D_5
D_6(8)	—	(14)－D_6
D_7(9)	—	(16)－D_7
－ACK(10)	—	(48)－REQ
BUSY(11)	—	(42)－MSG
PE(12)	—	(36)－BSY
SELCT(13)	—	(46)－C/D
－AUTFD(14)	—	(44)－SEL
－ERROR(15)	—	(50)－I/O
－IPRT(16)	—	(32)－ATN
－SLIN(17)	—	(40)－RST
GND(18～25)	—	(25以外の奇数ピン，20,22,24,28,30,34)
DSUB25J		フラット・ケーブル用コネクタ

(b) HDD側がフラット・ケーブル・タイプの場合

クト個所が2個所

▶ 10Mバイトずつふたつのパーテーションに分割

という操作を行っています.

〈CR〉が付いている部分が入力部分です.

SPCカードのI/Oアドレスはどこになっていますか? FDF0〈CR〉

フォーマットしたいドライブのSCSI-IDは何番ですか? 0〈CR〉

ロジカル・ユニット番号(ドライブの番号)は何番ですか? 0〈CR〉

物理フォーマットを行いますか?(Y/N) Y〈CR〉

セクタ・サイズをいくつにしますか?(256・512) 256〈CR〉

インターリーブ・ファクタは, いくつにしますか? 2〈CR〉

一度もフォーマットを行ったことがないドライブですか?(Y/N) Y〈CR〉

このドライブの, シリンダ数はいくつですか? 615〈CR〉

ヘッド数はいくつですか? 4〈CR〉

書き込み位相補正(ライト・プリコンペ)を始めるシリンダはいくつですか?(不要な場合は, シリンダ数と同じ数) 256〈CR〉

書き込み電流補正(リデュースド・ライト・カレント)を始めるシリンダはいくつですか?(不要な場合は, シリンダ数と同じ数) 615〈CR〉

これでよいですか?(Y/N) Y〈CR〉

ディフェクトの個数はいくつですか? 2〈CR〉

ディフェクト 1: シリンダは? 100〈CR〉

ヘッドは? 0〈CR〉

インデックスからの距離(バイト)は? 6535〈CR〉

ディフェクト 2: シリンダは? 140〈CR〉

ヘッドは? 3〈CR〉

インデックスからの距離(バイト)は? 8008〈CR〉

ディフェクト 1: シリンダ 100 ヘッド 0 6535バイト

ディフェクト 2: シリンダ 140 ヘッド 3 8008バイト

これでよいですか? (Y/N) Y〈CR〉

ハードディスクの媒体検査を行いますか?(Y/N) Y〈CR〉

媒体検査を何回繰り返しますか? 1〈CR〉

パーティションをいくつにしますか?(1~12) 2〈CR〉

残りセクタ数は, 40326(20646912バイト)です.

パーティション 1 に何セクタ割り当てますか?(最大40326) 20163〈CR〉

残りセクタ数は, 20163(10323456バイト)です.

パーティション 2 に何セクタ割当てますか?(最大20163) 20163〈CR〉

パーティション 1: 20163 セクタ(10323456バイト)

パーティション 2: 20163 セクタ(10323456バイト)

残りセクタ:0

これでよいですか? (Y/N) Y〈CR〉

これで, フォーマットが終了します.

〈図12〉パラレル・インターフェース・レジスタの割り当て

•データ(×78H または ×7CH)

7	6	5	4	3	2	1	0

ライト:SCSIバス・データ出力(プリンタ出力データ)
リード:SCSIバス・データ入力(プリンタ入力データ)

•プリンタ・コントロール(×7AH または ×7EH)

7	6	5	4	3	2	1	0
未使用	未使用	未使用	IRQ Enable	RST(SLCT IN)	ATN(INIT)	SEL(AUTO FD XT)	ACK(STROBE)

•プリンタ・ステータス(×79H または ×7DH)

7	6	5	4	3	2	1	0
MSG(BUSY)	REQ(ACK)	BSY(PE)	C/D(SLCT)	I/O(ERROR)	未使用	未使用	未使用

Easy Hard

SCSIバスへのインターフェースを行うためには, なにがなんでもSCSIコントローラ・チップを使わなければならないわけではありません. 安易に汎用ロジックの8ビット・ラッチとバス・ドライバなどだけでも実現することができます.

そのような発想からパソコン通信のネット上で発表されたのがEasy Hardです.

もっともEasy Hardは夕陽郎というハンドル・ネームの方のアイデアで, IBM-PCコンパチのプリンタ・ポートへSCSIのハードディスクをケーブルだけで接続してしまおうというものです.

なぜ, このようなことが可能であるかというと, もともとIBM-PCのプリンタ・ポートは図10のようにデータが双方向の入出力が可能なように設計されているためで, これにプリンタの制御信号部分をSCSIの制御信号に割り当てることによって, SCSIのバス・フ

ェーズをソフトによって制御することができるというわけです.

このため，ハードの製作は**図11**の変換ケーブルを1本作るだけで済んでしまいます.

図12にパラレル・インターフェース・レジスタをSCSIとして使った場合の割り当てを示します(カッコ中は本来のプリンタ・ポートとしての割り当て).

● ドライバ

対象とするシステムは現在のところJ3100SS/SL/GX/SGT101，IBM-PC/AT，30Z-SXです．**リスト8〜リスト13**に全プログラム(ドライバ，フォーマット・プログラム)を示します．MAKEFILEを実行することで，.SYSと.EXEができます.

8086コードで書かれています．使用したアセンブラはMASM5.1です.

対象になるドライブはANSIの標準に準拠したSCSIドライブ(CCSが使えるドライブ)です.

NEC PC9801-55カードに接続できるドライブ，および富士通FM-R/TOWNSシリーズに接続できるドライブなどであればまったく問題なく使用できます.

ただし，SCSIバスのパリティをディセーブルにしなければなりません.

具体的には，例えばD3841/D5852などのNECのドライブの場合にはたくさん並んでいるディップ・スイッチの中のPARITYと書いてあるスイッチをOFFにします.

富士通のM1055Bではディップ・スイッチの5番をOFFにします．M2246SAならPと書いてあるジャンパを抜きます.

● ドライバの使用方法

ケーブルの製作が終わったら，まずプリンタ・ポートを双方向に設定します.

- J3100SS：
  「SetupJ」でポートを「プリンタ・出力専用」に設定します.
- J3100SL：
  「SetupJ」で「プリンタ接続：なし」を選択します.
- J3100GX：
  プリンタ・コネクタ脇のディップ・スイッチ2番をONにします.
- J3100SGT101：
  プリンタ・コネクタ脇のディップ・スイッチ4番をONにします.

つぎにHDDを接続してSFEH.EXEを実行します．PC/ATなどの漢字の出せない機械の場合には英語版SFEHE.EXEを使用してください.

SFEHはMS-Cで書いてあり，SCSIのハンドシェイクもすべてCで行っているので，実行時にSCSI.SYSがある必要はありません.

これが正常終了したら，ケーブルは多分正常なので，あとはCONFIG.SYSにSCSI.SYSを組み込めば終わりです．こちらは英語/日本語には関係ありません.

ターゲットID，ロジカル・ユニット番号(LUN)はCONFIG.SYSの中でパラメータとして渡せます.

速いCPUのためにリセット後のディレイ時間調整を入っており，

例： DEVICE=SCSI.SYS /I4 /L2 /W4
　　ID4，LUN2を選択，リセット後のウェイト4

と指定します.

パラメータを付けない場合のデフォルトのターゲットIDは0，ターゲットのLUNも0です．IDは0から6まで，LUNは0から3までを受け付けます.

リセット後のウェイト時間のデフォルトは2です．値は1から9までを受け付けます.

IとLとW以外のパラメータは受け付ません.

なお，起動時に論理ブロック0を読み込んでパーテーション情報などをセットしているので，起動時には必ずHDDがレディ状態になっている必要があります.

▶注意

この接続の場合，電源OFF時に一瞬バス・フェーズが不安定になることがあるようで，ドライブによってはいったん回転を停止してしまいます.

その場合，リセットをかけたときに一度タイム・アウトになり，イニシャライズ・エラーが出る場合があります．これは避けようがないので，あきらめて再度リセットをしてください.

リセットをせずにファイル・アクセスなどをする場合にはエラー表示のときにリトライを選択すればそのまま続行できると思います.

J3100SL/SSの場合，レジュームという厄介な機能があります.

電源切断中にHDDをつなぎ替えていない場合については対応していますが，違うHDDを接続した場合には必ず一度リセットを行ってください．場合によってはFATを破壊することがあると思います(そんな恐ろしいこと自分ではとても試せない).

このドライバは256/512セクタ両用になっています．起動時に勝手に判断するのでユーザはフォーマット時にどちらかを選んでください.

ターゲットIDというのはSCSIコントローラ(ドライブ)のSCSIバス上でのIDです．7はパソコンが使うので，0から6までです.

LUNというのは，SCSIコントローラとハードディスク・ドライブ間での機番です．ST506インターフェースのHDDの場合，ドライブの機番が1から4まで選べますが，これが1のときがLUN=0です．SCSI内蔵型HDDの場合は必ず0になります.

このドライバ単体ではマルチ・ターゲット(複数台のHDD)はサポートしていませんが，SCSI ID か LUN を変更して CONFIG.SYS に登録すれば複数台接続も可能です．若干メモリの無駄になりますが，1ドライブあたり 1.5K バイト位なので我慢できる範囲だと思います．

しかし，ケーブル長が延びることになるので，避けた方が無難です．MS-DOS はデータ化けについてはノーチェックです．

パーテーションは 12 まで切れます．それぞれ最大 32M バイトまでです．ドライバ・イニシャライズのときにドライブの論理セクタ 0(DD セクタ)を読み込んで，それから判断するのでブートの際に必ずデバイスがレディになっている必要があります．

くりかえしますが，前述のように MS-DOS には，「標準で組み込まれているデバイスよりも大きなセクタ・サイズのブロック・デバイスは追加できない」という闇の掟があります．

ですから，英語 DOS などでは 512 バイト/セクタ以上にできないので，32M バイト以上のパーテーションが切れません(256 しか使えない OS9 よりはましかも？)．

もっとも日本語 DOS しか絶対に使用しない，というのならば，1024 バイト/セクタに見せることもできるので 64M バイトのパーテーションも可能ですが……．

DD セクタの構成はつぎの通りです．(論理ブロック 0)

2 バイト：パーテーションの個数

4 バイト＊12：それぞれのパーテーションの開始する論理ブロック・アドレス

14 バイト：未使用

16 バイト＊12：パーテーションごとのＢＰＢ情報

合計 256 バイト

やはり，ソフト転送しかできないので非常に遅くなります．70K バイト/s 程度しか出ません．M1055B の場合，インターリーブは 8 が最適でした．

SPC の場合には J3100SS でも 300K バイト/s 近く出ますから，約 1/4 の実力といったところです．

簡単さを取るか，性能を取るかの問題で，J3100SS ならこの位でも十分かなあ？とは思っています(小さいファイルのアクセスならほとんど変わらないし)．

純正 PC/AT(286/8MHz)でもなぜか 100K バイト/s 位です．不思議だなあ？

● Easy Hard のフォーマット・プログラム

SPC ドライバとデータ互換があるので，こちらでフォーマットしたドライブは SPC カードでそのまま使えます．

以下に手順を示しますが，基本的な操作は SPC と同じです．

M1055B＋D3126 で，初めてフォーマットを行う場合の操作例を示します．ここでは

▶ SCSI ID，LUN は 0，ディフェクト個所が 2 個所ある

▶セクタ・サイズは 512 バイトに設定

▶媒体検査は 2 回行う

▶ 10M バイトずつふたつのパーテーションに分割

という操作を行っています．

<CR>が付いている部分が入力部分です．

フォーマットしたいドライブの SCSI-ID は何番ですか？　0<CR>

ロジカル・ユニット番号(ドライブの番号)は何番ですか？　0<CR>

物理フォーマットを行いますか？　(Y/N)　Y<CR>

セクタ・サイズをいくつにしますか？(256・512)　512<CR>

インターリーブ・ファクタは，いくつにしますか？　8<CR>

一度もフォーマットを行ったことがないドライブですか？(Y/N)　Y<CR>

このドライブの，シリンダ数はいくつですか？　615<CR>

ヘッド数はいくつですか？　4<CR>

書き込み位相補正(ライト・プリコンペ)を始めるシリンダはいくつですか？(不要な場合は，シリンダ数と同じ数)　256<CR>

書き込み電流補正(リデュースド・ライト・カレント)を始めるシリンダはいくつですか？(不要な場合は，シリンダ数と同じ数)　615<CR>

これでよいですか？　(Y/N)　Y<CR>

ディフェクトの個数はいくつですか？　2<CR>

ディフェクト　1:　シリンダは？　100<CR>
ヘッドは？　0<CR>

インデックスからの距離(バイト)は？　6535<CR>

ディフェクト　2:　シリンダは？　140<CR>
ヘッドは？　3<CR>

インデックスからの距離(バイト)は？　8008<CR>

ディフェクト　1: シリンダ 100 ヘッド 0　6535 バイト

ディフェクト　2: シリンダ 140 ヘッド 3　8008 バイト

これでよいですか？　(Y/N)　Y<CR>

ハードディスクの媒体検査を行いますか？　(Y/N)　Y<CR>

媒体検査を何回繰り返しますか？　2<CR>

パーティションをいくつにしますか？　(1～12) 2<CR>

残りセクタ数は，40326(20646912 バイト)です．

パーティション 1 に何セクタ割り当てますか？(最大

40326)　　　　　　　　　　　　　　　　20163<CR>
残りセクタ数は，20163(10323456バイト)です．
パーティション 2 に何セクタ割り当てますか？(最大
20163)　　　　　　　　　　　　　　　　20163<CR>
パーティション 1: 20163 セクタ(10323456 バイト)
パーティション 2: 20163 セクタ(10323456 バイト)
残りセクタ：0
これでよいですか？ (Y/N)　　　　　　　　Y<CR>

これで，フォーマットが終了します．

● 注意点など

速度的には，SPCなどを使った専用ハードウェアより遅いのですが，ケーブル1本だけでハードディスクが接続できてしまうため，インターフェース・ボードなどを内蔵できない東芝のダイナブックなどのノート型パソコンなどでは，一部には必需品とさえいわれているようです．

注意として，電気レベル的には本来SCSIバスを駆動することを考えていないプリンタ・ポートを使うのですからしかたがないのですが，SCSIのデータ・パリティがないので接続するドライブはそのチェックを行わないように，設定できるドライブあるいは無視するドライブを選ぶ必要があります．

また，パリティ・チェックを行っていないためケーブルを極力短くすることも大切です．

〈リスト2〉SPC170.ASM

```
        PAGE    66,132
;
;MB89352 (SCSIプロトコルコントローラ) ドライバ
;copyright by 鶴蔵
;
;V1.0   July.25,1989    ハード転送ルーティンの高速化 (8バイトずつ転送を行う)
;V1.1   July.28,1989    ハード転送ルーティンの高速化、転送中のフェーズチェック
;V1.2   Aug. 02,1989                            ルーティン追加
;
;V1.3   Aug. 03,1989    リクエストセンスコマンドパラメータをANSI対応に変更
;                                       パーティション数を12個までに変更
;
;V1.4   Sep. 08,1989    ストリングス命令を使用して、ハード転送ルーティンを高速
;                                                                       化
;
;V1.45  Nov. 07,1989    J3100SL/SSのレジューム機能に対応
;V1.5   Mar. 01,1990    256/512Bytes/Sct兼用に書換え、config.sysのパラメータを
;                                               サポート
;
;V1.6   Jul. 28,1990    1024Bytes/Sct、リムーバブルメディアをサポート
;V1.7   Oct. 17,1990    Open/Close/RMビットを追加
;
;コマンド、ステータス、メッセージはソフト転送
;リード/ライトルーティンのデータ転送フェーズのみ
;ハード転送 (但しDMAは使用しない)
;
;物理フォーマットは256・512・1024バイト/セクタ
;
;シングルターゲット (ターゲットID、LUNはConfig．sysで定義)
;
;パーティションはドライブあたり12まで、パーティションサイズはおのおの
;最大32 (256/512Bytes/Sct時) /64 (1024Bytes/Sct時) MBまで
;
IFDEF   I186
                .186
ENDIF
                .SALL
                INCLUDE SPC.INC

CODE    SEGMENT PUBLIC
                ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
;
;       デバイスヘッダー
;
;
DSKDEV  LABEL   WORD
                DD              -1                      ;最終デバイス、にしておく
                DW              2800H           ;Block Device,Non FAT-ID,Open/Close/RM
                DW              STRATEGY        ;ストラテジルーティンエントリ
                DW              DSK_INT         ;インタラプトルーティンエントリ

                PAGE
;
;       ワークエリア
;
PTRSAV  DD              0       ;リクエストヘッダアドレス退避用
TMPPHA  DB              ?       ;Phase change の検出用
;
;
;       SCSIコマンドバッファ (リード/ライト時用)
;
SCSI    SCSIWORK        <       >

RW_FLG          DB      ?       ;0の時リード、1の時ライト
VRFY_FLG        DB      ?       ;1の時ベリファイオン (未使用)
UNIT_NO         DB      ?       ;ユニット番号
SEC_CNT         DW      ?       ;セクタカウント
TFRMODE         DB      ?       ;転送モード(ソフト転送時0,ハード転送で0ffh)
SSIZ            DB      ?       ;セクタサイズ(256の時1,512の時2,1024の時4が入る)
INITBIT         DB      ?       ;イニシエータIDビット
TAGTBIT         DB      ?       ;ターゲットIDビット
LUN                     DB      ?       ;ロジカルユニットナンバ
;
;       DDセクタエリア
;       ここに起動時に論理ブロック0を読み込んで来る
;
;       ここは、パーティション情報等が入る (64バイト)
;
DDSCT           DDPARAMS        <       >
;
;       ここからパーティション毎のBPBが入る (16*12バイト)
;
```

〈リスト1〉MAKEFILE (SPC用)

```
spc186.sys : spc170.asm spc.inc
        masm /DI186 spc170.asm,spc186.obj;
        link spc186;
        del spc186.obj
        exe2bin spc186 spc186.sys
        del spc186.exe

spc86.sys : spc170.asm spc.inc
        masm spc170.asm,spc86.obj;
        link spc86;
        del spc86.obj
        exe2bin spc86 spc86.sys
        del spc86.exe

spc186mo.sys : spc170.asm spc.inc
        masm /DI186 /DMO spc170.asm,spc186mo.obj;
        link spc186mo;
        del spc186mo.obj
        exe2bin spc186mo spc186mo.sys
        del spc186mo.exe

spc86mo.sys : spc170.asm spc.inc
        masm /DMO spc170.asm,spc86mo.obj;
        link spc86mo;
        del spc86mo.obj
        exe2bin spc86mo spc86mo.sys
        del spc86mo.exe

scsi.obj : scsi.c spc.h
        cl -Zp -c scsi.c

sform.exe : sform.c scsi.obj spc.h
        cl -Zp -F 4000 -Fesform.exe sform.c scsi.obj

sformmo.exe : sformmo.c scsi.obj spc.h
        cl -Zp -F 4000 -Fesformmo.exe sformmo.c scsi.obj
```

```
DSKBPB0         BPBTYP          <       >       ;パーティション0
DSKBPB1         BPBTYP          <       >       ;パーティション1
DSKBPB2         BPBTYP          <       >       ;パーティション2
DSKBPB3         BPBTYP          <       >       ;パーティション3
DSKBPB4         BPBTYP          <       >       ;パーティション4
DSKBPB5         BPBTYP          <       >       ;パーティション5
DSKBPB6         BPBTYP          <       >       ;パーティション6
DSKBPB7         BPBTYP          <       >       ;パーティション7
DSKBPB8         BPBTYP          <       >       ;パーティション8
DSKBPB9         BPBTYP          <       >       ;パーティション9
DSKBPB10        BPBTYP          <       >       ;パーティション10
DSKBPB11        BPBTYP          <       >       ;パーティション11

;
;       リクエストセンス専用のワークエリア（リトライをする関係上）
;
SENSDAT         DB      768             DUP(?)  ;センスデータ格納場所
REQS            SCSIWORK        <       >

INIT_TBL:       ;BPB配列へのポインタ
                DW              DSKBPB0         ;Unit #0
                DW              DSKBPB1         ;Unit #1
                DW              DSKBPB2         ;Unit #2
                DW              DSKBPB3         ;Unit #3
                DW              DSKBPB4         ;Unit #4
                DW              DSKBPB5         ;Unit #5
                DW              DSKBPB6         ;Unit #6
                DW              DSKBPB7         ;Unit #7
                DW              DSKBPB8         ;Unit #8
                DW              DSKBPB9         ;Unit #9
                DW              DSKBPB10        ;Unit #10
                DW              DSKBPB11        ;Unit #11

CMDADDR         DW              ?               ;SCSIコマンドパケットアドレス
BUFADDR         DW      2 DUP(?)        ;転送先のアドレス（ES：DI）

IFDEF   MO              ;MO－Discの場合

DEV_NO          DB              ?               ;デバイス番号
MEDIA_CHGD      DB      12 DUP(1)       ;メディアが交換されたかどうかのフラッグ

ENDIF

                PAGE
;
;       各処理へのジャンプテーブル
;
JMPTBL  LABEL   WORD
                DW              DSK_INIT        ; 0 Init
                DW              MEDIA_CHK       ; 1 Media check
                DW              GET_BPB         ; 2 Build BPB
                DW              CMD_ERR         ; 3 IOCTL Input
                DW              DSK_READ        ; 4 Input
                DW              EXIT            ; 5 Non-Destructive input no wait
                DW              EXIT            ; 6 Input status
                DW              EXIT            ; 7 input flush
                DW              DSK_WRT         ; 8 Output
                DW              DSK_WRT         ; 9 Output with verify
                DW              EXIT            ;10 Output status
                DW              EXIT            ;11 Output flush
                DW              EXIT            ;12 IOCTL Output
                DW              EXIT            ;13 Device Open
                DW              EXIT            ;14 Device Close
                DW              RM_MEDIA        ;15 Removable Media
                DW              EXIT            ;16 Output Until Busy
                DW              EXIT            ;17 Reserved
                DW              EXIT            ;18 Reserved
                DW              EXIT            ;19 Generic IOCTL
                DW              EXIT            ;20 Reserved
                DW              EXIT            ;21 Reserved
                DW              EXIT            ;22 Reserved
                DW              EXIT            ;23 Get Drive Map
                DW              EXIT            ;24 Set Drive Map
;
;*
;*      ストラテジルーティン
;****************
STRATP  PROC    FAR
STRATEGY:
                MOV             WORD PTR CS:[PTRSAV],BX
                MOV             WORD PTR CS:[PTRSAV+2],ES
                RET
STRATP  ENDP

        PAGE
;
;*
;*      インタラプトルーティン
;****************
DSK_INT:
                PUSH    SI              ;全レジスタセーブ
                PUSH    AX
                PUSH    CX
                PUSH    DX
                PUSH    DI
                PUSH    BP
                PUSH    DS
                PUSH    ES
                PUSH    BX

DSK_INT1:
                LDS             BX,CS:[PTRSAV]  ;リクエストヘッダアドレスを得る
                MOV             AL,[BX.UNIT]    ;ユニットコード
                MOV             AH,[BX.MEDIA]   ;メディアディスクリプタ
                MOV             CX,[BX.COUNT]   ;セクタカウント
                MOV             DX,[BX.START]   ;スタートブロック
                XCHG    AX,DI
                MOV             AL,[BX.CMD]             ;コマンドコード
                CMP             AL,16
                JA              CMD_ERR                 ;16以上はエラーにしている（手抜き）
                XOR             AH,AH
                SHL             AX,1
                MOV             SI,OFFSET JMPTBL
                ADD             SI,AX
                XCHG    AX,DI
                LES             DI,[BX.TRANS]   ;転送先アドレスを得る
                PUSH    CS
                POP             DS                      ;DS←CS
                JMP             [SI]                    ;各処理に分岐する
        PAGE
;*
;*      終了ルーティン
;****************
CMD_ERR:                                                ;コマンドエラーの場合
                MOV             AL,3                    ;ALに、エラーコード3をセット
ERR_EXIT:
                MOV             AH,10000001B    ;エラービットとDoneビットをセット
                JMP             SHORT EXIT1

EXITP   PROC    FAR
EXIT:                                                   ;ノーエラー
                MOV             AH,00000001B    ;Doneビットのみセット
EXIT1:
                LDS             BX,CS:[PTRSAV]  ;リクエストヘッダアドレスを得る
                MOV             [BX.STATUS],AX  ;ステータスバイトをセット

                POP             BX                      ;レジスタ復帰
                POP             ES
                POP             DS
                POP             BP
                POP             DI
                POP             DX
                POP             CX
                POP             AX
                POP             SI
                RET
EXITP   ENDP
        PAGE
;****************
;*      コマンド処理ルーティン
;****************

;*
;*      コマンド1：メディアチェック
;****************
MEDIA_CHK:                                              ;メディアが交換されたかのチェック
```

〈リスト2〉SPC170.ASM（つづき）

```
IFDEF   MO                      ;MO－Discの場合
        MOV     DEV_NO,AL                   ;デバイス番号をセーブ

        CALL    TESTREADY

        MOV     AL,BYTE PTR REQS.STAT
        CMP     AL,02H                      ;Check condition ?
        JNE     MEDIA_CHKEND

        CALL    REQSENS                 ;Media Changed の検出

        MOV     AL,BYTE PTR SENSDAT+12
        CMP     AL,28H                      ;Media changed ?
        JE      READ_DDSCT
        CMP     AL,29H                      ;Power-On Reset ?
        JE      READ_DDSCT

        JMP     MEDIA_CHKEND            ;else Media not changed

;メディアが交換されているので、DDセクタの読み込みを行う

READ_DDSCT:
        MOV     REQS.CMDS,C$REDSEC
        MOV     AL,BYTE PTR LUN
        MOV     REQS.SECH,AL        ;ブロックアドレス0から
        MOV     REQS.SECM,00H
        MOV     REQS.SECL,00H
        MOV     REQS.BLOCK,01H      ;1セクタだけ読み込む
        MOV     REQS.CNTL,00H
        LEA     AX,REQS
        PUSH    CS
        POP     ES                          ;ES←CS
        LEA     DI,DDSCT

        MOV     TFRMODE,00H             ;マニュアル転送を選択
        CALL    HDCMD

        MOV     AL,0FFH                 ;ユニット番号0～11の
        LEA     DI,MEDIA_CHGD       ;Media changed flagをセット
        MOV     CX,12
        REP     STOSB
MEDIA_CHKEND:
        LEA     BX,MEDIA_CHGD           ;そのユニット番号で「交換されている」
        MOV     AL,DEV_NO
        MOV     AH,0
        ADD     BX,AX                       ;を既に返したかどうかをチェック
        MOV     AL,BYTE PTR [BX]
        CMP     AL,01H                      ;01hか、0ffhが入っている
        JE      NOT_CHGD                    ;01hの場合は交換されていない
CHGD:
        MOV     BYTE PTR [BX],01H       ;そのユニット番号のMedia Changed Flag
                                            ;をクリア
                                            ;「交換されている」を返す
NOT_CHGD:
        LDS     BX,[PTRSAV]
        MOV     BYTE PTR [BX.TRANS],AL  ;「交換されてない」を返す
        JMP     EXIT
ELSE                            ;Hard－Diskの場合

        LDS     BX,[PTRSAV]
        MOV     BYTE PTR [BX.TRANS],1   ;常に「交換されてない」を返す
        JMP     EXIT

ENDIF

;*
;*  コマンド2：ビルドBPB
;******************
GET_BPB:                                    ;BPBを返す
        MOV     AH,0
        SHL     AX,1
        LEA     DI,INIT_TBL
        ADD     DI,AX
        MOV     BX,[DI]
        MOV     AH,BYTE PTR [BX]+10     ;FAT－IDバイト
        MOV     SI,BX                       ;BPBのアドレス
```

```
        LDS     BX,[PTRSAV]                     ;リクエストヘッダのアドレスを得る
        MOV     [BX.MEDIA],AH               ;メディアディスクリプタ
        MOV     [BX.COUNT],SI               ;BPBのアドレスをセット
        MOV     [BX.COUNT+2],CS
        JMP     EXIT

;*
;*  コマンド4：インプット
;****************
DSK_READ:                                           ;ディスクリードルーティン
        MOV     RW_FLG,0                            ;「リードモード」をセット
        MOV     VRFY_FLG,0
        JMP     RW_COMMON

;*
;*  コマンド8：アウトプット
;****************
DSK_WRT:                                    ;ディスクライトルーティン
        MOV     RW_FLG,1                    ;「ライトモード」をセット
        MOV     VRFY_FLG,0                  ;ベリファイはOFF
        JMP     RW_COMMON
;*
;*  コマンド9：アウトプット&ベリファイ
;********************
DSK_WRTV:                                   ;ライト&ベリファイ
        MOV     RW_FLG,1                    ;「ライトモード」をセット
        MOV     VRFY_FLG,1                  ;ベリファイはON
        JMP     RW_COMMON                   ;と言っても、現在ベリファイは
                                                ;サポートしてない。
;
;  リード／ライトの共通ルーティン
;
RW_COMMON:
        JCXZ    RWEXIT                  ;セクタカウントが0の時は何もしない
        JMP     RWC1
RWEXIT:
        JMP     EXIT                        ;美しくないなあ。。。
RWC1:   MOV     UNIT_NO,AL              ;ユニットナンバ、セクタ数を
        MOV     SEC_CNT,CX                  ;セーブ

        CMP     RW_FLG,0                    ;リード?
        JNE     WRT
                                                ;リードルーティン
        MOV     SCSI.CMDS,C$REDSEC      ;SCSIのリードコマンドをセット
        JMP     RW10
WRT:
        MOV     SCSI.CMDS,C$WRTSEC      ;SCSIのライトコマンドをセット
;
RW10:
        MOV     AH,0                        ;パーティション分のセクタ
        SHL     AX,1                        ;オフセットの計算
        SHL     AX,1
        LEA     BX,DDSCT.SCTOFS         ;DDセクタに書かれていた値と
        ADD     BX,AX                       ;リクエストヘッダの値を加算する

        ADD     DX,WORD PTR [BX]        ;セクタオフセットを加算
        MOV     AL,0
        ADC     AL,BYTE PTR [BX+2]

        CMP     BYTE PTR SSIZ,01H       ;セクタサイズ=256の場合は
        JNE     RW20
        SHL     CX,1                        ;セクタカウント*=2
        SHL     DX,1                        ;セクタアドレス*=2
        RCL     AL,1
RW20:
        OR      AL,BYTE PTR LUN
        MOV     SCSI.SECH,AL            ;SCSIコマンドパケットの作成
        MOV     SCSI.SECM,DH
        MOV     SCSI.SECL,DL
        MOV     SCSI.BLOCK,CL
        MOV     SCSI.CNTL,0

        LEA     AX,SCSI                     ;AXにコマンドパケットアドレス
        MOV     TFRMODE,0FFH            ;転送モードはブラインド転送
        CALL    HDCMD                   ;SCSIハンドルルーティンを呼ぶ

        JC      RWERR                       ;セレクションタイムアウト
```

〈リスト2〉SPC170.ASM（つづき）

```
        MOV     AL,SCSI.STAT
        CMP     AL,0                    ;ノーエラーならそのままリターン
        JE      RWRTN

        CALL    REQSENS                 ;チェックの場合はリクエストセンスをして
        JC      RWERR
        MOV     AL,SENSDAT+2
        CMP     AL,1                    ;リカバードエラーは無視する
        JE      RWRTN
        CMP     AL,6                    ;リセットコンディションの時はリードライト
        JE      RWRETRY                 ;出来てないからリトライ

        CLC
        JMP     RWERR                   ;それ以外は本当のエラー
RWRETRY:                                ;リトライする
        LEA     AX,SCSI
        MOV     TFRMODE,0FFH
        CALL    HDCMD

        JC      RWERR
        MOV     AL,SCSI.STAT
        CMP     AL,0
        JNE     RWERR                   ;リトライはしない
RWRTN:                                  ;正常終了
        LDS     BX,[PTRSAV]             ;リクエストヘッダアドレスを得る
        MOV     [BX.COUNT],CX           ;転送されたセクタ数をセット
        JMP     EXIT
RWERR:                                  ;異常終了
        JNC     RWE1
        MOV     AL,E$NOTRDY             ;ALにエラーコードをセットして
        JMP     RWERET
RWE1:
        TEST    RW_FLG,01H
        JNE     RWE2
        MOV     AL,ESREAD
        JMP     RWERET
RWE2:
        MOV     AL,ESWRITE
RWERET:
        JMP     ERR_EXIT                ;ERR＿EXITにジャンプ
        RET

;*
;*  コマンド15：リムーバブルメディアチェック
;************************
RM_MEDIA:
IFDEF   MO                              ;removable mediaの場合は、Busyビットをクリア
        MOV     AX,0100H
ELSE
        MOV     AX,0300H                ;HDDの場合は、Busyビットをセット
ENDIF
        JMP     EXIT1


;
;
;       テストユニットレディ
;
TESTREADY:
        PUSH    DI                      ;転送アドレスは保存しておく
        PUSH    ES

        MOV     REQS.CMDS,C$TSTRDY      ;専用バッファアドレスにコマンド
        MOV     AL,BYTE PTR LUN
        MOV     REQS.SECH,AL            ;パケットをセット
        MOV     REQS.SECM,00H           ;
        MOV     REQS.SECL,00H           ;
        MOV     REQS.BLOCK,00H
        MOV     REQS.CNTL,00H

        LEA     AX,REQS
        LEA     DI,SENSDAT              ;

        PUSH    CS
        POP     ES                      ;CS -> ES
```

```
        MOV     TFRMODE,0               ;ソフト転送モードを選択
        CALL    HDCMD

        POP     ES
        POP     DI
        RET
;
;
;       リクエストセンス
;
REQSENS:
        PUSH    DI                      ;転送アドレスは保存しておく
        PUSH    ES

        MOV     REQS.CMDS,C$REQSEN      ;専用バッファアドレスにコマンド
        MOV     AL,BYTE PTR LUN
        MOV     REQS.SECH,AL            ;パケットをセット
        MOV     REQS.SECM,00H           ;この後でコマンドリトライをする
        MOV     REQS.SECL,00H           ;事があるのでバッファを分けてある
        MOV     REQS.BLOCK,128
        MOV     REQS.CNTL,00H

        LEA     AX,REQS
        LEA     DI,SENSDAT              ;ここにセンスデータが格納される

        PUSH    CS
        POP     ES                      ;CS -> ES

        MOV     TFRMODE,0               ;ソフト転送モードを選択
        CALL    HDCMD

        POP     ES
        POP     DI
        RET
        PAGE
;
;       SCSIハンドリングルーティン
;       AXにコマンドパケットアドレス
;       ES:DIに転送バッファアドレス
;       AX,BX,CX,DX:破壊、ES,DI:保存
;
HDCMD:
        MOV     CMDADDR,AX              ;アドレスのセーブ
        MOV     BUFADDR,DI
        MOV     BUFADDR+2,ES
        PUSH    CS
        POP     ES

        GETP    P$SCTL                  ;SPCコントロールレジスタを読む
        TEST    AL,10H                  ;アービトレーションイネーブルビットを調べる
        JNE     HDCMD0                  ;セットしてあればレジュームはされていない

        CALL    SPC_INIT                ;一度レジュームされているので、SPCの
                                        ;レジスタを再度初期化する
HDCMD0:
        PUTPI   P$PCTL,00H              ;セレクションフェーズ
        GETP    P$BDID
        OR      AL,BYTE PTR TAGTBIT
        PUTP    P$TEMP                  ;ターゲット、イニシエータIDを送出
        PUTPI   P$TCH,0FH               ;セレクションタイムアウトカウンタセット
        PUTPI   P$TCM,46H
        PUTPI   P$TCL,04H
        PUTPI   P$INTS,00H
        PUTPI   P$SCMD,S$SELECT

HDCMD1:                                 ;インタラプトを待つ
        GETP    P$INTS
        AND     AL,14H
        JE      HDCMD1

        CMP     AL,10H                  ;旨くセレクション出来たらしい
        JE      HDCMDCMD

        MOV     AL,E$NOTRDY             ;セレクションタイムアウト
        PUSH    AX
        PUSH    CX
        PUTPI   P$SCMD,10H              ;バスリセットを送出
        MOV     CX,100
```

```
        LOOP    $
        PUTPI   P$SCMD,00H
        POP             CX
        POP             AX
        MOV             CX,100
EREX1:  PUSH    CX
        MOV             CX,10000
        LOOP    $
        POP             CX
        LOOP    EREX1
        JMP             HDCMDERR

HDCMDCMD:               ;コマンドフェーズ
        GETP    P$PSNS          ;REQが1になるまで待つ
        TEST    AL,80H
        JE              HDCMDCMD

        AND             AL,T$MSG+T$CD+T$IO
        CMP             AL,T$CD         ;コマンドフェーズかチェック
        JNE             HDCMDCMD

        PUTP    P$PCTL          ;転送フェーズレジスタのセット

        MOV             DI,CMDADDR      ;DI←コマンドパケットアドレス

        CALL    MANTFR          ;マニュアル転送する
        JNC             HDCMDTFR

        JMP     HDCMDERR

HDCMDTFR:               ;トランスファーフェーズ
        GETP    P$PSNS          ;REQが1になるまで待つ
        TEST    AL,80H
        JE              HDCMDTFR

        TEST    AL,T$MSG+T$CD   ;ステータスフェーズの場合もあるので
        JNE             HDCMDSTS                ;チェックする

        AND             AL,T$MSG+T$CD+T$IO
        PUTP    P$PCTL                  ;転送フェーズレジスタのセット

        MOV             DI,BUFADDR              ;DI←データバッファアドレス
        MOV             ES,BUFADDR+2    ;ES←データバッファセグメント

        TEST    TFRMODE,0FFH    ;ハード転送モード?
        JE              HDMAN

        CALL    HARDTFR                 ;ハード転送
        JMP             HDCT1
HDMAN:
        CALL    MANTFR                  ;マニュアル転送
HDCT1:
        PUSH    CS
        POP             ES
HDCMDSTS:               ;ステータスフェーズ
        GETP    P$PSNS
        TEST    AL,80H                  ;REQが1になるまで待つ
        JE              HDCMDSTS

        AND             AL,T$MSG+T$CD+T$IO
        CMP             AL,T$CD+T$IO
        JNE             HDCMDSTS

        PUTP    P$PCTL                  ;

        MOV             BX,CMDADDR
        LEA             DI,[BX].STAT
        CALL    MANTFR                  ;ステータスバイトを得る

HDCMDMES:               ;メッセージインフェーズ
        GETP    P$PSNS
        TEST    AL,80H                  ;REQが1になるまで待つ
        JE              HDCMDMES

        AND             AL,T$MSG+T$CD+T$IO
        CMP             AL,T$MSG+T$CD+T$IO
        JNE             HDCMDMES
        PUTP    P$PCTL                  ;転送フェーズレジスタのセット

        PUSH    CS
        POP             ES

        MOV             BX,CMDADDR
        LEA             DI,[BX].MESS
        CALL    MANTFR                  ;メッセージバイトを得る

HDCMDBF:                ;バスフリーフェーズ
        GETP    P$PSNS
        TEST    AL,08H                  ;ビジーが0になるまで待つ
        JNE             HDCMDBF

        GETP    P$INTS
        AND             AL,20H
        JNE             HDCMDBF1

        PUTP    P$INTS
        MOV             AL,E$NOTRDY             ;デバイスビジー
        JMP             HDCMDERR

HDCMDBF1:               ;ディスコネクト
        GETP    P$PCTL
        AND             AL,7FH                  ;バスフリー割り込み禁止
        PUTP    P$PCTL
        GETP    P$INTS                  ;割り込みソースのクリア
        PUTP    P$INTS
        GETP    P$SCTL

        MOV             DI,BUFADDR              ;バッファアドレスの復帰
        MOV             ES,BUFADDR+2
        CLC                     ;正常終了
        RET

HDCMDERR:               ;異常終了
        MOV             DI,BUFADDR
        MOV             ES,BUFADDR+2
        STC
        RET
    PAGE
;
;       マニュアル転送ルーティン
;       フェーズが変化するまで転送する
;       バッファアドレスはES:DI
;       AX,DX:破壊、ES,DI:保存
;
MANTFR:
        PUSH    DI
MANTFR10:
        GETP    P$PSNS
        TEST    AL,80H                  ;REQが1になるまで待つ
        JE              MANTFR10

        GETP    P$PCTL                  ;DATA INかOUTかチェック
        TEST    AL,T$IO
        JE              MANOUT

MANIN:                  ;DATA IN フェーズ
        PUTPI   P$SCMD,S$SETACK ;ACKを返して

        MOV             DX,P$PSNS
MANI1:  IN      AL,DX                   ;REQが0になるまで待つ
        TEST    AL,80H
        JNE             MANI1

        GETP    P$TEMP
        STOSB                           ;データを貰う

        PUTPI   P$SCMD,S$RESETACK

        MOV             DX,P$PSNS
MANI2:  IN      AL,DX

        TEST    AL,88H                  ;バスフリー?
        JE              MANTFREXIT              ;の時はEXIT
```

〈リスト2〉SPC170.ASM（つづき）

```
        TEST    AL,80H                  ;次のＲＥＱが来るまで待つ
        JE              MANI2

        AND             AL,T$MSG+T$CD+T$IO      ;今度のフェーズを得る
        MOV             AH,AL

        GETP    P$PCTL
        AND             AL,T$MSG+T$CD+T$IO
        CMP             AH,AL                   ;今までと同じ？
        JE              MANIN                   ;ならもう１バイト

        JMP             MANTFREXIT              ;終りました！

MANOUT:                         ;ＤＡＴＡ　ＯＵＴフェーズ
        MOV             AL,ES:[DI]              ;データを出力する
        PUTP    P$TEMP
        PUTPI   P$SCMD,S$SETACK ;ＡＣＫを返す

        MOV             DX,P$PSNS
MANO1:  IN      AL,DX                   ;ＲＥＱが０に戻るまで待つ
        TEST    AL,80H
        JNE             MANO1

        PUTPI   P$SCMD,S$RESETACK
        INC             DI                              ;バッファポインタを進める

        MOV             DX,P$PSNS
MANO2:  IN      AL,DX

        TEST    AL,88H                  ;バスフリー？
        JE              MANTFREXIT              ;ならば終了する

        TEST    AL,80H                  ;次のＲＥＱが来るまで待つ
        JE              MANO2

        AND             AL,T$MSG+T$CD+T$IO      ;今度のフェーズを得る
        MOV             AH,AL

        GETP    P$PCTL
        AND             AL,T$MSG+T$CD+T$IO
        CMP             AH,AL                   ;今までと同じフェーズ？
        JE              MANOUT                  ;ならばもう１バイト

MANTFREXIT:
        POP             DI                              ;終了
        CLC
        RET

;
;       ハードウェア転送ルーティン
;       現在は、データリード／ライトコマンド時のみ使用
;       データバッファアドレスはＥＳ：ＤＩ
;       AX,CX,DX:破壊、ES,DI,SI:保存
;
HARDTFR:
        PUSH    DI
        PUSH    SI
        MOV             BX,CMDADDR              ;ＳＣＳＩリクエストヘッダから
        MOV             AL,[BX].BLOCK           ;転送すべきブロック数を得て
        CMP             BYTE PTR SSIZ,01H       ;セクタサイズ＝２５６の場合は
        JE              HARDTFR5                ;そのまま
        SHL             AL,1                    ;５１２、１０２４の時は２倍する
        CMP             BYTE PTR SSIZ,02H       ;
        JE              HARDTFR5                ;
        SHL             AL,1                    ;１０２４の時は、もう２倍する
HARDTFR5:
        PUTP    P$TCM                   ;でそれをＳＰＣの転送語数カウ
        PUTPI   P$TCH,0                 ;ンタにセットして
        PUTP    P$TCL
        PUTPI   P$SCMD,84H              ;ハード転送命令を出す（ＤＭＡ無し）

        PUSH    DS
        PUSH    ES                              ;DS←ES
        POP             DS
        CLD
        MOV             BX,P$SSTS
HARDTFR10:                      ;転送が始まるまで待つ
        MOV             DX,BX
        IN              AL,DX
        AND             AL,0F0H
        CMP             AL,0B0H
        JNE             HARDTFR10
HARDTFR15:
        GETP    P$PCTL                          ;データイン／アウトをチェックする
        SHR             AL,1
        JNC             HARDOUT0
HARDIN:
        MOV             DX,BX                   ;ＦＩＦＯバッファのチェック
        IN              AL,DX                   ;データが８バイト入るまでループ
        SHR             AL,1
        SHR             AL,1
        JC              HARDIN2
        GETP    P$PSNS
        SHR             AL,1
        SHR             AL,1
        JC              HARDTFR20               ;フェーズが変わっていれば終了

        JMP             HARDIN
HARDIN1:
        MOV             DX,BX                   ;ＤＸ←Ｐ＄ＳＳＴＳ
        IN              AL,DX                   ;ＦＩＦＯバッファのチェック
        SHR             AL,1                    ;データが８バイト入るまでループ
        SHR             AL,1
        JNC             HARDIN
HARDIN2:
        MOV             DX,P$DREG
IFDEF   I186
        MOV             CX,8
        REP             INSB
ELSE
        IN              AL,DX
        STOSB                                   ;データを得る
        IN              AL,DX
        STOSB
        IN              AL,DX
        STOSB
        IN              AL,DX
        STOSB
        IN              AL,DX
        STOSB
        IN              AL,DX
        STOSB
        IN              AL,DX
        STOSB
        IN              AL,DX
        STOSB
ENDIF
        MOV             DX,BX                   ;ＤＸ←Ｐ＄ＳＳＴＳ
        IN              AL,DX                   ;ＦＩＦＯバッファのチェック
        SHR             AL,1                    ;データが８バイト入るまでループ
        SHR             AL,1
        JC              HARDIN2

        JMP     HARDIN1
HARDOUT0:
        MOV             SI,DI
HARDOUT:                        ;データアウトの場合
        MOV             DX,BX                   ;ＤＸ←Ｐ＄ＳＳＴＳ
        IN              AL,DX                   ;ＦＩＦＯバッファのチェック
        SHR             AL,1
        JC              HARDOUT2                ;バッファが完全に空くまでループ

        GETP    P$PSNS
        SHR             AL,1
        SHR             AL,1
        JC              HARDTFR20               ;フェーズが変わっていれば終了

        JMP             HARDOUT
```

〈リスト2〉SPC170.ASM（つづき）

```
HARDOUT1:
        MOV     DX,BX                   ;ＤＸ←Ｐ＄ＳＳＴＳ
        IN      AL,DX                   ;ＦＩＦＯバッファのチェック
        SHR     AL,1
        JNC     HARDOUT                 ;バッファが完全に空くまでループ
HARDOUT2:
        MOV     DX,P$DREG
IFDEF   I186
        MOV     CX,8
        REP     OUTSB
ELSE
        LODSB                           ;データを送出
        OUT     DX,AL
        LODSB
        OUT     DX,AL
        LODSB
        OUT     DX,AL
        LODSB
        OUT     DX,AL
        LODSB
        OUT     DX,AL
        LODSB
        OUT     DX,AL
        LODSB
        OUT     DX,AL
        LODSB
        OUT     DX,AL
ENDIF
        MOV     DX,BX                   ;ＤＸ←Ｐ＄ＳＳＴＳ
        IN      AL,DX                   ;ＦＩＦＯバッファのチェック
        SHR     AL,1
        JC      HARDOUT2                ;バッファが完全に空くまでループ
        JMP     HARDOUT1
HARDTFR20:
        POP     DS
        GETP    P$SSTS          ;終了ステータスをチェックする
        CMP     AL,85H
        JNE     HARDTFRERR
HARDTFREXIT:
        POP     SI
        POP     DI                              ;正常終了
        CLC
        RET
HARDTFRERR0:
        POP     DS
        JMP     HARDTFREXIT
HARDTFRERR:
        POP     SI
        POP     DI                              ;異常終了
        STC
        RET
    PAGE
;＊ＳＰＣの内部レジスタのイニシャライズルーティン
;＊通常は、Ｉｎｉｔルーティンより一回呼ばれるだけだが、Ｊ３１００ＳＬ／ＳＳ
;＊の場合はレジュームされるたびにＣａｌｌされる。
;＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
;    全レジスタ保存
;
SPC_INIT:
        PUSH    AX
        PUSH    DX
        PUTPI   P$SCTL,10011000B        ;ＳＰＣリセット
        PUTPI   P$BDID,I$INITID         ;自分のＩＤ（７）をセット
        PUTPI   P$SCMD,00H
        PUTP    P$TMOD
        PUTP    P$PCTL
        PUTP    P$TEMP
        PUTP    P$TCH
        PUTP    P$TCM
        PUTP    P$TCL
        PUTP    P$PSNS
        PUTPI   P$SCTL,00011000B        ;ＳＰＣリセットクリア
```

```
        POP     DX
        POP     AX
        RET

;＊
;＊  イニットルーティン（デバイス登録時に１度だけ呼ばれる）
;＊  改版してちょっと長くなったが、起動後は切り放されるから良しとしよう
;＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
DSK_INIT:                                       ;まずＳＣＳＩバスのイニシャライズ
        LEA     DX,MSG                          ;Ver.,etc.を表示
        MOV     AH,09H
        INT     21H
                                                ;ＳＣＳＩアドレスの初期値を設定
        MOV     BYTE PTR INITBIT,I$INITBIT      ;イニシエータＩＤ＝７
        MOV     BYTE PTR TAGTBIT,I$TAGTBIT      ;デフォルトのTarget ID＝０
        MOV     BYTE PTR LUN,I$LUN              ;デフォルトのLogical Unit No.＝０

        LDS     BX,[PTRSAV]                     ;リクエストヘッダアドレスを得る

        MOV     DI,WORD PTR [BX.COUNT]
        MOV     ES,WORD PTR [BX.COUNT+2]
        MOV     AL,BYTE PTR [BX.COUNT+4]        ;最初のドライブ番号を得る
        PUSH    CS
        POP     DS
        MOV     BYTE PTR STTDRV,AL
        CALL    PRMCHK                          ;Config.sysのparameterをチェック
                                                ;リードキャパシティ実行
DINIT0:
        MOV     SCSI.CMDS,C$RDCAP
        MOV     AL,BYTE PTR LUN
        MOV     SCSI.SECH,AL
        MOV     SCSI.SECM,00H
        MOV     SCSI.SECL,00H
        MOV     SCSI.BLOCK,00H
        MOV     SCSI.CNTL,00H
        MOV     SCSI.EXT1,00H
        MOV     SCSI.EXT2,00H
        MOV     SCSI.EXT3,00H           ;PMI bit false
        MOV     SCSI.EXT4,00H           ;flag and link bit false

        LEA     AX,SCSI
        PUSH    CS
        POP     ES                                      ;ＥＳ←ＣＳ
        LEA     DI,DDSCT                                ;ＤＩ←ＤＤＳＣＴ

        MOV     TFRMODE,00H                             ;マニュアル転送を選択
        CALL    HDCMD
        JNC     DINIT01
        JMP     DIERR
DINIT01:
        CMP     SCSI.STAT,00H                   ;ステータスチェック
        JE      DINIT0END                       ;ノーエラー

        CALL    REQSENS                         ;センスデータを得る
        JNC     DINIT02
        JMP     DIERR
DINIT02:
        MOV     AL,BYTE PTR SENSDAT+2
        CMP     AL,01H                          ;リカバードエラーは無視する
        JE      DINIT0END
        CMP     AL,06H                          ;リセットコンディションだともう１度
        JE      DINIT0
DINIT0END:
        MOV     AL,BYTE PTR DDSCT+6             ;セクタサイズを得る
        MOV     BYTE PTR SSIZ,AL

DINIT1:                                         ;ＤＤセクタの読み込み
        MOV     SCSI.CMDS,C$REDSEC
        MOV     AL,BYTE PTR LUN
        MOV     SCSI.SECH,AL                    ;ブロックアドレス０から
        MOV     SCSI.SECM,00H
        MOV     SCSI.SECL,00H
        MOV     SCSI.BLOCK,01H                  ;１セクタだけ読み込む
        MOV     SCSI.CNTL,00H

        LEA     AX,SCSI
        PUSH    CS
        POP     ES                                      ;ＥＳ←ＣＳ
```

```
		LEA		DI,DDSCT

		MOV		TFRMODE,00H		;マニュアル転送を選択
		CALL	HDCMD
		JC		DIERR
		CMP		SCSI.STAT,00H		;ステータスチェック
		JE		DIEX1			;ノーエラー

		CALL	REQSENS			;センスデータを得る
		JC		DIERR
		MOV		AL,BYTE PTR SENSDAT+2
		CMP		AL,01H			;リカバードエラーは無視する
		JE		DIEX1
		CMP		AL,06H			;リセットコンディションだともう1度
		JE		DINIT1
DIERR:
		LEA		DX,MSGERR		;失敗のメッセージを表示
		MOV		AH,09H
		INT		21H

		PUSH	DS
		MOV		AL,0
		LDS		BX,[PTRSAV]		;リクエストヘッダアドレスを得る
		MOV		[BX.MEDIA],AL		;ドライブ数←0

		MOV		WORD PTR [BX.TRANS],OFFSET DSK_INIT
		MOV		WORD PTR [BX.TRANS+2],CS

		MOV		WORD PTR [BX.COUNT],OFFSET INIT_TBL
		MOV		WORD PTR [BX.COUNT+2],CS
								;BPB配列のポインタ
		POP		DS

		STC
		MOV		AL,ESNOTRDY		;エラーコードのセット
		JMP		ERR_EXIT
DIEX1:
		LEA		DX,MSGCMP		;成功のメッセージを表示
		MOV		AH,09H
		INT		21H

		MOV		CH,0
		MOV		CL,BYTE PTR DDSCT.DRVNUM	;DDセクタから、パーティションの
		MOV		DL,BYTE PTR STTDRV		;数を得る
		ADD		DL,'A'
		LEA		BX,DSKBPB0+9
DIEX2:
		PUSH	DX
		MOV		AH,02H			;ドライブ番号の表示
		INT		21H
		MOV		DL,':'
		INT		21H
		MOV		AL,BYTE PTR [BX]
		MOV		AH,BYTE PTR SSIZ
		CMP		AH,04H			;1024Bytes/sectorの場合
		JE		DIEX3
		SHR		AL,1
DIEX3:
		MOV		AH,0
		ADD		AX,02H			;1MB以下を四捨五入
		SHR		AX,1
		AND		AX,0FEh			;AX = Drive_Capacity(MB) * 2
		PUSH	BX
		LEA		BX,CAPTABLE
		ADD		BX,AX
		MOV		DL,BYTE PTR [BX]
		MOV		AH,02H
		INT		21H
		MOV		DL,BYTE PTR [BX+1]
		INT		21H
		POP		BX
		LEA		DX,MSGCAP
		MOV		AH,09H
		INT		21H
		ADD		BX,16			;BPBアドレスを進める
		POP		DX
		INC		DL			;ドライブ番号+=1
```

```
		LOOP	DIEX2

		LEA		DX,MSGEND
		MOV		AH,09H
		INT		21H

		MOV		AL,BYTE PTR DDSCT.DRVNUM
		PUSH	DS
		LDS		BX,[PTRSAV]		;リクエストヘッダアドレスを得る
		MOV		[BX.MEDIA],AL		;ドライブ数←パーティションの数

		MOV		WORD PTR [BX.TRANS],OFFSET DSK_INIT
		MOV		WORD PTR [BX.TRANS+2],CS

		MOV		WORD PTR [BX.COUNT],OFFSET INIT_TBL
		MOV		WORD PTR [BX.COUNT+2],CS
								;BPB配列のポインタ
		POP		DS
		CLC						;エラー無し

		JMP		EXIT

;
;	起動パラメータのチェック
;	文字列のポインタはES:DI
;
PRMCHK:							;search '/' char.
		CALL	PRMGET1
		JC		PRMCHKEND		;if End of line
		CMP		AL,'/'
		JE		PRMCHK1
		JMP		PRMCHK			;次の文字
;
PRMCHK1:						;'/'の次の文字を得る
		CALL	PRMGET1
		JC		PRMCHKERR
		AND		AL,0DFH			;toupper(*s)
		CMP		AL,'I'
		JE		PRMCHKID		;'/I'の場合
		CMP		AL,'L'
		JE		PRMCHKLUN		;'/L'の場合
		JMP		PRMCHKERR

PRMCHKID:						;change target ID
		CALL	PRMGET1
		JC		PRMCHKERR
		SUB		AL,'0'			;if ID < 0 goto error
		JL		PRMCHKERR
		CMP		AL,6			;if ID > 6 goto error
		JA		PRMCHKERR
		MOV		AH,0
		MOV		CX,AX
		MOV		AL,01H
		CMP		CX,0
		JE		PRMID2
PRMID1:	SHL		AL,1
		LOOP	PRMID1
PRMID2:
		MOV		BYTE PTR TAGTBIT,AL
		JMP		PRMCHK
;
PRMCHKLUN:						;change LUN
		CALL	PRMGET1
		JC		PRMCHKERR
		SUB		AL,'0'			;if LUN < 0 goto error
		JL		PRMCHKERR
		CMP		AL,3			;if LUN > 3 goto error
		JA		PRMCHKERR
		MOV		CX,5
PRMLUN1:						;LUN = unit no. * 20H
		SHL		AL,1
		LOOP	PRMLUN1

		MOV		CS:BYTE PTR LUN,AL
		JMP		PRMCHK
;
PRMCHKERR:
		LEA		DX,MSGPRMERR		;パラメータ不正のメッセージを表示
		MOV		AH,09H
```

〈リスト2〉SPC170.ASM（つづき）

```
            INT         21H
PRMCHKEND:
            RET
PRMGET1:
            MOV         AL,ES:[DI]          ;search '/'
            INC         DI
            CMP         AL,0DH
            JE          PRMGETEND
            CMP         AL,0AH
            JE          PRMGETEND
            CLC
            RET
PRMGETEND:
            STC
            RET
;
; 起動時用のワークエリア
;
STTDRV  DB          ?

;
; 起動メッセージ
;
MSG         DB          0AH,0DH

IFDEF   MO
            DB          '  SPC Driver V1.7 for MO-Disc   '
ELSE
            DB          '  SPC Driver V1.7 for Hard-Disk '
ENDIF
            DB          '         Copyright by Tsuru-Zoh, Oct.17,1990',0AH,0DH
MSGEND  DB          0AH,0DH,'$'
MSGCMP  DB          '  Initialize Complete!    Drive ','$'
MSGERR  DB          '  Initialize Error.',0AH,0DH,'$'
MSGPRMERR:
            DB          '   Invalid parameter.',0AH,0DH,'$'
MSGCAP  DB          'MB  ','$'
CAPTABLE:
            DB          ' 0 1 2 3 4 5 6 7 8 9101112131415'
            DB          '16171819202122232425262728293031'
            DB          '32333435363738394041424344454647'
            DB          '48495051525354555657585960616263 64'
;
CODE    ENDS
        END
```

〈リスト3〉SPC.INC

```
;       SPCボードのI／Oアドレスが0300H番地以外の場合は、
;       ここを書き換えて下さい。
SPCBAS  EQU     0300H           ;SPCのベースアドレス

I$INITID        EQU     7       ;イニシエータID
I$INITBIT       EQU     80H

I$TAGTID        EQU     00      ;ターゲットID（デフォルト値0）
I$TAGTBIT       EQU     01H

I$LUN           EQU     00H     ;ロジカルユニット番号（デフォルト値0）

;
;       SPCのI／Oアドレス定義
;
P$BDID  EQU     SPCBAS+00H
P$SCTL  EQU     SPCBAS+01H
P$SCMD  EQU     SPCBAS+02H
P$TMOD  EQU     SPCBAS+03H
P$INTS  EQU     SPCBAS+04H
P$PSNS  EQU     SPCBAS+05H
P$SSTS  EQU     SPCBAS+06H
P$SERR  EQU     SPCBAS+07H
P$PCTL  EQU     SPCBAS+08H
P$MBC   EQU     SPCBAS+09H
P$DREG  EQU     SPCBAS+0AH
P$TEMP  EQU     SPCBAS+0BH
P$TCH   EQU     SPCBAS+0CH
P$TCM   EQU     SPCBAS+0DH
P$TCL   EQU     SPCBAS+0EH

;
;       SPCのコマンド定義
;
S$SELECT        EQU     20H
S$TRANSFER      EQU     80H
S$RESETACK      EQU     0C0H
S$SETACK        EQU     0E0H
S$RESET         EQU     10H

;
;       SCSIのコマンド定義
;
C$TSTRDY        EQU     00H
C$REZERO        EQU     01H
C$REQSEN        EQU     03H
C$FMTDRV        EQU     04H
C$REDSEC        EQU     08H
C$WRTSEC        EQU     0AH
C$SEEK          EQU     0BH
C$MODSEL        EQU     15H
C$MODSEN        EQU     1AH
C$RDCAP         EQU             25H

;
;       リクエストヘッダ構造体の定義
;
REQ_HEAD        STRUC
CMDLEN          DB      ?               ;コマンドのバイト数
UNIT            DB      ?               ;ユニット番号
CMD                     DB      ?       ;コマンド番号
STATUS          DW      ?               ;実行ステータス格納場所
                        DB      8 DUP(?)        ;システムリザーブ
MEDIA           DB      ?               ;メディアディスクリプタ
TRANS           DD      ?               ;転送バッファアドレス
COUNT           DW      ?               ;転送セクタ数
START           DW      ?               ;開始ブロック番号
REQ_HEAD        ENDS

;
;       SCSIコマンドパケット構造体の定義
;
SCSIWORK        STRUC
CMDS            DB      ?       ;SCSIのコマンドコード
SECH            DB      ?       ;セクタ番号
SECM            DB      ?       ;
SECL            DB      ?       ;
BLOCK           DB      ?       ;転送セクタ数
```

〈リスト3〉 SPC.INC（つづき）

```
CNTL        DB    ?      ;
EXT1        DB    ?      ;グループ1コマンド用拡張
EXT2        DB    ?      ;
EXT3        DB    ?      ;
EXT4        DB    ?      ;
STAT        DB    ?      ;実行ステータス格納場所
MESS        DB    ?      ;実行メッセージ格納場所
SCSIWORK    ENDS

;
;       DDセクタ領域のバイト定義
;
DDPARAMS    STRUC
DRVNUM      DB    ?            ;このドライブのパーティションの数
DDRSV1      DB    ?
SCTOFS      DD    12 DUP (?)   ;パーティションの開始するセクタ番号
DDRSV2      DB    14 DUP(?)
DDPARAMS    ENDS

;
;       MS-DOSのBPB構造体の定義
;       1ドライブあたり16バイト確保している
;
BPBTYP      STRUC
SCTSIZ      DW    ?      ;セクタサイズ
ALLSIZ      DB    ?      ;アロケーションユニットあたりのセクタ数
RSVSCT      DW    ?      ;リザーブセクタ数
FATCPY      DB    ?      ;FATのコピー数
DIRENT      DW    ?      ;ルートディレクトリのエントリ数
SCTVOL      DW    ?      ;このユニットの総セクタ数
MEDIAD      DB    ?      ;メディアディスクリプタバイト
FATNO       DW    ?      ;FATのセクタ数
BPBRSV      DB    3 DUP(?)
BPBTYP      ENDS

;
;       エラーコードの定義
;
ESWP        EQU   00H    ;WRITE PROTECT
ESERRUNIT   EQU   01H    ;ILLEGAL UNIT
ESNOTRDY    EQU   02H    ;NOT READY
ESERRCMD    EQU   03H    ;ILLEGAL UNIT
ESCRC       EQU   04H    ;CRC ERROR
ESREQLEN    EQU   05H    ;ERROR @ REQ. LENGTH
ESSEEK      EQU   06H    ;SEEK ERROR
ESMEDIA     EQU   07H    ;MEDIA ERROR
ESSECTOR    EQU   08H    ;SECTOR NOT FOUND
ESPAPER     EQU   09H    ;PAPER OUT
ESWRITE     EQU   0AH    ;WRITE ERROR
ESREAD      EQU   0BH    ;READ ERROR
ESERR       EQU   0CH    ;GLOBAL ERROR
ESCHG       EQU   0FH    ;MEDIA CHANGE ERROR

;
;       バスステータスのビット定義
;
TSMSG       EQU   04H
TSCD        EQU   02H
TSIO        EQU   01H

;
;       レジスタ構造体の定義
REGSTER STRUC
AHm         DB    ?
ALm         DB    ?
BHm         DB    ?
BLm         DB    ?
CHm         DB    ?
CLm         DB    ?
DXm         DW    ?
DIm         DW    ?
DSm         DW    ?
REGSTER ENDS

;       I/Oポートアクセスのマクロ定義
;
CLRP    MACRO   ADDR
        MOV     DX, ADDR
        IN      AL, DX
        MOV     AL, 0
        OUT     DX, AL
        ENDM

GETP    MACRO   ADDR
        MOV     DX, ADDR
        IN      AL, DX
        ENDM

PUTP    MACRO   ADDR
        MOV     DX, ADDR
        OUT     DX, AL
        ENDM

PUTPI   MACRO   ADDR, DATA
        MOV     AL, DATA
        MOV     DX, ADDR
        OUT     DX, AL
        ENDM

PUTPW   MACRO   ADDR
        MOV     DX, ADDR
        OUT     DX, AL
        MOV     AL, AH
        OUT     DX, AL
        ENDM

GETPW   MACRO   ADDR
        PUSH    CX
        MOV     DX, ADDR
        IN      AL, DX
        MOV     CL, AL
        IN      AL, DX
        MOV     AH, AL
        MOV     AL, CL
        POP     CX
        ENDM
```

〈リスト4〉 SCSI.C

```
/* Format program for MB89352( SPC ) on J-3100
 Converted from 63C09E Assembler ( Written by Angie ) by MAL5
 SCSI handling routines
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "spc.h"

extern unsigned scsibase;
extern int sctsiz;

/* SCSI BUS-Reset */
reset()
{
    long l, t;

    printf( "YnSCSIバスリセットを実行中です。10秒お待ち下さい。Yn" );

    outp( scsibase + S_SCMD, RESET );
    for( l = 0L; l < 3000L; l++ ){
    }
    outp( scsibase + S_SCMD, 0x00 );
    l = time( (long*)0 );
    while( ( t = ( time( (long*)0 ) - l ) ) < 10L ){
        printf( "%ldYr", t );
    }
    printf( "     Yn" );
}


verifyunit( id, lun, maxblock )
int id, lun;
long maxblock;
{
    long l, lt, le, badblock;
    unsigned char buf[ 256 ];
    int times, i;

    printf( "YnYn最大ブロック番号は、%ldです。Yn", maxblock );
    printf( "不良ブロックがある場合は、自動的に交替処理を行います。Yn" );
    printf( "Yn媒体検査を何回繰り返しますか？  " );
    scanf( "%d", &times );
    putchar( 'Yn' );
    for( i = 0; i < times; i++ ){
        printf( "媒体検査  :  %d 回目Yn", i + 1 );
        for( l = 0L; l < maxblock; l += 1024L ){
            printf( "ブロック %ldYr", l );
            lt = l;
            le = ( ( lt + 1024L ) > maxblock )? maxblock - l: 1024L;
            while( verify( id, lun, lt, le, buf ) ){
                badblock = 65536L * buf[ 4 ] + 256L * buf[ 5 ] + buf[ 6 ];
                reassign( id, lun, badblock );
                le -= ( badblock - lt + 1 );
                lt = badblock + 1L;
                if( lt == ( l + 1024L ) ){
                    break;
                }
            }
        }
    }
    printf( "  媒体検査が終了しました。  Yn" );
}

makepart( id, lun, maxsct )
int id, lun;
long maxsct;
{
    static struct{
        int pn;
        long sctofs[ 20 ];
    }ddsct;

    static struct{
        int         ssiz;
        char        allsiz;
```

〈リスト4〉 SCSI.C（つづき）

```
        int         rsvsct;
        char        fatcpy;
        int         dirent;
        unsigned    sctvol;
        char        mediad;
        int         fatno;
    }bpb[ 20 ] = {
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
        { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }
    };

    int patno, i, j;
    unsigned vol;
    long restsct, logsct, l;
    char buf[ 1024 ];

    logsct = ( sctsiz == 256 ) ? 512L: (long)sctsiz;    /* 論理セクタサイズ */
    maxsct = maxsct / ( ( sctsiz == 256 ) ? 2L: 1L ) - 3L;
    do{
        restsct = maxsct;
        printf( "¥nディスクの総セクタ数は、%ld(%ldバイト)です。¥n",
            maxsct, maxsct * logsct );
        printf( "¥nパーティションを幾つにしますか？ （１～１２）  " );
        scanf( "%d", &patno );
        for( i = 0; i < patno; i++ ){
            bpb[ i ].ssiz = (int)logsct;
            do{
                printf( "¥n残りセクタ数は、%ld(%ldバイト)です。¥n", restsct, restsct * logsct );
                printf( "パーティション %d に何セクタ割当てますか？(最大%ld) ",
                i + 1, (restsct > 65535L )? 65535L: restsct );
                scanf( "%d", &vol );
            }while( vol > 65535 || vol > restsct );
            bpb[ i ].sctvol = vol;
            restsct -= vol;
            if( vol > 20480 ){
                bpb[ i ].allsiz = 4;
            }else if( vol > 10240 ){
                bpb[ i ].allsiz = 2;
            }else{
                bpb[ i ].allsiz = 1;
            }
            bpb[ i ].fatno = vol / bpb[ i ].allsiz * 2 / logsct + 1;
        }
        putchar( '¥n' );
        for( i = 0; i < patno; i++ ){
            printf( "パーティション %3d: %uセクタ(%ldバイト)¥n",
             i + 1, bpb[ i ].sctvol, logsct * bpb[ i ].sctvol );
        }
        printf( "残りセクタ：%ld¥n", restsct );
        printf( "これで良いですか？ (Y/N)  " );
        scanf( "%s", buf );
    }while( *buf != 'y' && *buf != 'Y' );

    printf( "¥nＦＡＴとディレクトリの初期化を行っています。¥n" );
    restsct = 3L;
    for( i = 0; i < patno; i++ ){
        printf( "パーティション %3d ¥n", i + 1 );
        for( j = 0; j < 1024; j++ ){
            buf[ j ] = 0;
        }
        for( l = 0; l < 2L * bpb[ i ].fatno + 33L; l++ ){
            swrite( id, lun, restsct + l, buf );
        }
        buf[ 0 ] = 0xf8;
        buf[ 1 ] = 0xff;
        buf[ 2 ] = 0xff;
        swrite( id, lun, restsct + 1L, buf );
        swrite( id, lun, restsct + 1L + bpb[ i ].fatno, buf );
        memcpy( &( buf[ 3 ] ), "SPC V1.6", 8 );
        memcpy( &( buf[ 11 ] ), &( bpb[ i ] ), sizeof( bpb[ i ] ) );
        swrite( id, lun, restsct, buf );
        restsct += bpb[ i ].sctvol;
    }

    restsct = 3L;
    ddsct.pn = patno;
    for( i = 0; i < 12; i++ ){
        ddsct.sctofs[ i ] = restsct;
        restsct += bpb[ i ].sctvol;
    }
    memcpy( &( buf[ 0 ] ), &ddsct, sizeof( ddsct ) );
    for( i = 0; i < 12; i++ ){
        memcpy( &( buf[ 64 + i * 16 ] ), &( bpb[ i ] ), sizeof( bpb[ i ] ) );
    }
    swrite( id, lun, 0L, buf );

}

/* Verify */
verify( id, lun, start, range, buf )
int id, lun;
long start, range;
unsigned char *buf;
{
    unsigned char v[ 10 ], sensbuff[ 128 ];
    int sens;

    v[ 0 ] = 0x2f;
    v[ 1 ] = lun << 5;
    v[ 6 ] = v[ 9 ] = 0x00;

    v[ 2 ] = ( start >> 24 ) & 0x000000ffL;
    v[ 3 ] = ( start >> 16 ) & 0x000000ffL;
    v[ 4 ] = ( start >> 8 ) & 0x000000ffL;
    v[ 5 ] = start & 0x000000ffL;
    v[ 7 ] = ( range >> 8 ) & 0x000000ffL;
    v[ 8 ] = range & 0x000000ffL;

    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        sens = requestsense( id, lun, buf );
        if( sens == 1 || sens == 3 ){
            return( 1 );
        }else{
            printsens( buf );
        }
    }
    return( 0 );
}

/* Reassign block */
reassign( id, lun, badblock )
int id, lun;
long badblock;
{
    unsigned char v[ 6 ], buf[ 128 ];

    printf( "ブロック%ld が不良です。交替処理を行います。¥n", badblock );

    v[ 0 ] = 0x07;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    buf[ 0 ] = buf[ 1 ] = buf[ 2 ] = 0x00;
    buf[ 3 ] = 4;
    buf[ 4 ] = ( badblock >> 24 ) & 0x000000ffL;
    buf[ 5 ] = ( badblock >> 16 ) & 0x000000ffL;
    buf[ 6 ] = ( badblock >> 8 ) & 0x000000ffL;
    buf[ 7 ] = badblock & 0x000000ffL;

    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, buf ) ){
            printsens( buf );
        }
    }
}
```

```
/* Test Unit ready */
testready( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], buf[ 80 ];
    int i;

    v[ 0 ] = C_TSTRDY;
    v[ 1 ] = ( lun << 5 );
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;

    dma = 0;    /* not DMA mode */

    return ( hdcmd( id, v, buf ) );
}

/* read capacity */
readcapacity( id, lun, buf )
int id, lun;
unsigned char *buf;
{
    unsigned char  v[ 10 ], sensbuf[ 256 ];

    v[ 0 ] = C_RDCAP;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;   /*logical block address is 0 */
    v[ 6 ] = v[ 7 ] = 0x00;
    v[ 8 ] = 0x00;              /* PMI bit false */
    v[ 9 ] = 0x00;              /* flag and link false */

    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}


/* Write 1 Sector */
swrite( id, lun, lsn, buf )
int id, lun;
long lsn;   /* logical sector number */
unsigned char *buf; /* data buffer */
{
    unsigned char v[ 6 ];
    unsigned char sensbuf[ 256 ];

    if( sctsiz == 256 ){
        lsn *= 2L;
    }
    v[ 0 ] = C_WRTSEC;
    v[ 4 ] = ( ( sctsiz == 256 )? 2: 1 );
    v[ 5 ] = 0x00;
    v[ 1 ] = ( lun << 5 ) + ( lsn >> 16 ) & 0x00ff;
    v[ 2 ] = ( lsn >> 8 ) & 0x00ff;
    v[ 3 ] = lsn & 0x00ff;
    dma = 0;    /* write 256 bytes */
    scsimode = 0x0b88;
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}


requestsense( id, lun, buf )
int id, lun;
unsigned char *buf;
{
    /* check error, do request sense. */
    unsigned char v[ 6 ];

    v[ 0 ] = C_REQSEN;
    v[ 1 ] = lun << 5;    /* set Lun */
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 255;
    dma = 0x00;
    hdcmd( id, v, buf );
    return( buf[ 2 ] );
}

printsens( buf )
unsigned char *buf;
{
    printf( "Sense key = %02X, Extend sense code = %02X¥n", buf[ 2 ], buf[ 12 ] );
    dump( buf );
    exit( 99 );
}


/* output command to HDD */
hdcmd( id, sc, buf )
int id;
unsigned char *sc, *buf;
{
    int treg;   /* temp. reg. */
    int idb;

    idb = 0x0001 << id;

    /* selection phase */
    outp( scsibase + S_PCTL, 0x00 );
    treg = inp( scsibase + S_BDID );
    outp( scsibase + S_TEMP, idb | treg );  /* output target & init. ID */
    outp( scsibase + S_TCH, 0x0f ); /* Set timeout Counter */
    outp( scsibase + S_TCM, 0x46 );
    outp( scsibase + S_TCL, 0x04 );

    outp( scsibase + S_INTS, 0x00 );
    outp( scsibase + S_SCMD, SELECT );
    while( !( ( treg = inp( scsibase + S_INTS ) ) & 0x14 ) ){
        /* wait IRQ */
    }
    if( treg != 0x10 ){
        printf( " Device Busy at selection phase.¥n " );
        exit( 99 );
    }

    /* command phase */
    do{
        while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ){
            /* wait until REQ = 1 */
        }
        treg &= ( MSG + CD + IO );
    }while( treg != CD );
    outp( scsibase + S_PCTL, treg | 0x80 ); /* bus free IRQ disable */
    mantfr( sc );

    /* data transfer phase */
    while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ){
        /* wait until REQ = 1 */
    }
    if( !( treg & ( MSG | CD ) ) ){ /* status phase? */
        /* no, data phase. */
        treg &= ( MSG | CD | IO );
        treg |= 0x80;   /* bus free IRQ enable */
        outp( scsibase + S_PCTL, treg );
        mantfr( buf );
    }

    /* status phase */
    do{
        while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ){
            /* wait until REQ = 1 */
        }
        treg &= ( MSG | CD | IO );
    }while( treg != ( CD | IO ) );
```

〈リスト4〉 SCSI.C（つづき）

```
	treg |= 0x80;
	outp( scsibase + S_PCTL, treg );	/* bus free IRQ enable */
	tfrsub( &sttbuf ); /* get Status byte */

	/* message phase */
	do{
		while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ){
			/* wait until REQ = 1 */
		}
		treg &= ( MSG | CD | IO );
	}while( treg != ( MSG | CD | IO ) );
	treg |= 0x80;
	outp( scsibase + S_PCTL, treg );	/* Bus free IRQ enable */
	tfrsub( &msgin );	/* get message byte */

	/* bus free phase */
	while( inp( scsibase + S_PSNS ) & 0x08 ){
		/* wait until BUSY = 0 */
	}
	if( !( treg = ( inp( scsibase + S_INTS ) & 0x20 ) ) ){
		outp( scsibase + S_INTS, treg );
		printf( "Device Busy at bus free phase.¥n" );
		exit( 99 );
	}
	/* disconnect */
	treg = inp( scsibase + S_PCTL ) & 0x7f;
	outp( scsibase + S_PCTL, treg );	/* bus free interrupt disable */
	treg = inp( scsibase + S_INTS );
	outp( scsibase + S_INTS, treg );	/* reset interrupt source */
	inp( scsibase + S_SCTL );
	treg = sttbuf;
	if( treg & 0x01 ){
		/* parity error */
		printf( "parity error occured.¥n" );
		exit( 99 );
	}else if( !( treg & 0x02 ) ){
		/* no error */
		return( 0 );
	}else{
		return( 1 );
	}
}

/* manual Data transfer routine */
mantfr( buf )
unsigned char *buf;
{
	int treg;	/* temporary register */

	do{
		tfrsub( buf++ );
		while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ){
			/* wait until REQ = 1 */
		}
		treg &= ( MSG | CD | IO );
	}while( treg == ( inp( scsibase + S_PCTL ) & ( MSG | CD | IO ) ) );
		return( 0 );
	}

	/* sub routine: 1 byte transfer */
	tfrsub( buf )
	char *buf;
	{
		int treg;	/* temp. reg. */

		while( !( inp( scsibase + S_PSNS ) & 0x80 ) ){
			/* wait until REQ = 1 */
		}
		if( inp( scsibase + S_PCTL ) & IO ){
			/* I/O IN */
			outp( scsibase + S_SCMD, SETACK );
			while( inp( scsibase + S_PSNS ) & 0x80 ){
				/* wait until REQ = 0 */
			}
			*buf = inp( scsibase + S_TEMP );
		}else{
			/* I/O OUT */
			outp( scsibase + S_TEMP, *buf );
			outp( scsibase + S_SCMD, SETACK );
			while( inp( scsibase + S_PSNS ) & 0x80 ){
				/* wait until REQ = 0 */
			}
		}
		outp( scsibase + S_SCMD, RESETACK );
}

/* SPC Initialize */
init(){
	int i;

	outp( scsibase + S_BDID, INITID );	/* controller ID = 7 */
	outp( scsibase + S_SCMD, 0x00 );
	outp( scsibase + S_TMOD, 0x00 );
	outp( scsibase + S_PCTL, 0x00 );
	outp( scsibase + S_TEMP, 0x00 );
	outp( scsibase + S_TCH,  0x00 );
	outp( scsibase + S_TCM,  0x00 );
	outp( scsibase + S_TCL,  0x00 );
	outp( scsibase + S_PSNS, 0x00 );
	outp( scsibase + S_SCTL, 0x98 );
	for( i = 0; i < 16; i++ ){
		/* for Delay */
	}
	outp( scsibase + S_SCTL, 0x18 );
}

dump( buf )
unsigned char *buf;
{
	int i, j;
	unsigned d;

	printf( "      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  0123456789ABCDEF¥n" );

	for( i = 0; i < 256; i += 16 ){
		printf( "%04X: ", i );
		for( j = 0; j < 16; j++ ){
			printf( "%02X ", buf[ i + j ] );
		}
		for( j = 0; j < 16; j++ ){
			d = buf[ i + j ];
			if( ( d >= ' ' ) && ( d < 0x7f ) ){
				putchar( d );
			}else{
				putchar( '.' );
			}
		}
		putchar( '¥n' );
	}
}
```

〈リスト5〉 SPC.H

```
/* Defs */

/* SPC Register */

#define S_BDID 0x00
#define S_SCTL 0x01
#define S_SCMD 0x02
#define S_TMOD 0x03
#define S_INTS 0x04
#define S_PSNS 0x05
#define S_SDGC 0x06
#define S_SSTS 0x06
#define S_SERR 0x07
#define S_PCTL 0x08
#define S_MBC  0x09
#define S_DREG 0x0a
#define S_TEMP 0x0b
#define S_TCH  0x0c
#define S_TCM  0x0d
#define S_TCL  0x0e

/* SPC Command */

#define SELECT   0x20
#define TRANSFER 0x80
#define RESETACK 0xC0
#define SETACK   0xE0
#define RESET    0x10

/* transfer Mode */

#define MSG 0x04
#define CD  0x02
#define IO  0x01

/* Controler command */

#define C_TSTRDY 0x00
#define C_REZERO 0x01
#define C_REQSEN 0x03
#define C_FMTDRV 0x04
#define C_REDSEC 0x08
#define C_WRTSEC 0x0A
#define C_SEEK   0x0B
#define C_MODSEL 0x15
#define C_MODSEN 0x1a
#define C_RDCAP 0x25

/* SCSI ID */

#define INITID 7
#define INITBIT 0x80
#define TAGTID 0
#define TAGTBIT 0x01

/* DMAC Register */
/* use DMAC Ch.1 */

#define D_ADDR      0x0002  /* c.1 address register */
#define D_COUNT     0x0003  /* ch.1 count register */
#define D_STATS     0x0008
#define D_CMND      0x0008
#define D_RQST      0x0009
#define D_MASK      0x000a
#define D_MODE      0x000b
#define D_CLR       0x000c
#define D_ALLMASK   0x000f
#define D_BANK      0x0083


/* Global Variables */

/* 6 bytes length command table */
typedef struct{
    unsigned char   cmds,   /* Op Code */
                    sech,   /* LSN High */
                    secm,   /* LSN Mid. */
                    secl,   /* LSN Low  */
                    block,  /* block no. */
                    ctrl    /* control byte */;
}SCSICMD;

/* data buffer for system */
char sysbuf[ 2048 ];

char sttbuf;    /* buffer for status byte */
char msgin; /* buffer for message byte */
char ints;  /* interrupt status */
int dma;
int dmamode;
int scsimode;
```

〈リスト6〉 SFORM.C

```
/* Format program for MB89352( SPC ) on J-3100
 Converted from 63C09E Assembler ( Written by Angie ) by MAL5

*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "spc.h"

unsigned scsibase;

int sctsiz;

main()
{
    unsigned char buf[ 512 ];
    int i, j, id, lun, inl, patno;
    long l, maxsct, stt;

    printf( "ＳＰＣカード用　ＨＤＤイニシャライズユーティリティＶ１．６¥n" );
    printf( "                    Copyright by Tsuru-Zo Jul.28.1990¥n¥n" );
    printf( "  ＊＊注意＊＊　実行すると、今ＨＤＤに入っているデータは全て破壊されます。¥n¥n" );
    printf( "      途中で中断したい時は、Ctrl-Cを押して下さい。¥n¥n" );

    printf( "ＳＰＣカードのＩ／Ｏアドレスはどこになっていますか？　" );
    scanf( "%x", &scsibase );

    printf( "¥nフォーマットしたいドライブのＳＣＳＩ－ＩＤは何番ですか？ " );
    scanf( "%d", &id );
    printf( "¥nロジカルユニット番号（ドライブの番号）は何番ですか？ " );
    scanf( "%d", &lun );

    init();
    reset( id, lun );

    printf( "      セクタサイズ、インタリーブ値を変更する場合は物理フォー¥n" );
    printf( "      マットを行って下さい。¥n" );
    printf( "¥n  物理フォーマットを行いますか？  (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        physformat( id, lun );
    }

    readcapacity( id, lun, buf );
    if( ( buf[ 6 ] != 1 ) && ( buf[ 6 ] != 2 ) && ( buf[ 6 ] != 4 ) ){
        printf( "¥n¥n＊＊＊物理セクタサイズが２５６、５１２、１０２４バイト以外になっています。¥n" );
        printf( "一度、物理フォーマットを行って下さい。¥n" );
        exit( 1 );
    }
    if( sctsiz == 0 ){
        sctsiz = buf[ 6 ] * 256;
    }
    if( ( buf[ 6 ] * 256 ) != sctsiz ){
        printf( "¥n¥n＊＊＊選択されたセクタサイズと物理セクタサイズが異なっています。¥n" );
        printf( "選択されたセクタサイズ：%d    物理セクタサイズ：%d¥n",
            sctsiz, buf[ 6 ] * 256 );
        sctsiz = buf[ 6 ] * 256;
        printf( "無視して処理を続行しますか？  (Y/N)" );
        scanf( "%s", buf );
        if( ( *buf != 'y' ) && ( *buf != 'Y' ) ){
            exit( 1 );
        }
    }
    maxsct = 65536L * buf[ 1 ] + 256L * buf[ 2 ] + buf[ 3 ] + 1L;

    printf( "¥n¥n¥nハードディスクの媒体検査を行いますか？ (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        verifyunit( id, lun, maxsct );
    }
    makepart( id, lun, maxsct );

    printf( "¥nフォーマットが完了しました。¥n" );
    printf( "Config.sysにＳＰＣ８６／１８６．ＳＹＳを登録し、リセットをして下さい。¥n" );
}
```

```
/* physical format */
physformat( id, lun )
int id,lun;
{
    int inl;
    char buf[ 32 ];

    printf( "¥n¥n     ＯＳ９と共用する場合は、２５６バイト／セクタとすると¥n" );
    printf( "    便利です。それ以外の時は５１２バイト／セクタにした方¥n" );
    printf( "    が容量が増える場合が多いです。また、５１２バイト／セ¥n" );
    printf( "    クタしか出来ないドライブもあります。¥n¥n" );
    do{
        printf( "セクタサイズを幾つにしますか？（２５６・５１２・１０２４）  " );
        scanf( "%d", &sctsiz );
    }while( ( sctsiz != 256 ) && ( sctsiz != 512 ) && ( sctsiz != 1024 ) );

    printf( "¥n¥n      コントローラにトラックバッファが有る場合は、必ずイ¥n" );
    printf( "    ンタリーブファクタを１にして下さい。¥n" );
    printf( "       トラックバッファが無い場合は、8086-10MHzでは３、80¥n" );
    printf( "    286-10MHz では２程度が適当な筈ですが、コントローラ、¥n" );
    printf( "    システムによって異なりますので各自実験してみて下さい。¥n" );
    printf( "¥nインタリーブファクタは、いくつにしますか？ " );
    scanf( "%d", &inl );

    printf( "¥n¥n       Ｙを押すと、ＨＤＤのシリンダ、ヘッド数、ディフェク¥n" );
    printf( "    ト情報入力画面になります。¥n" );
    printf( "       ＳＣＳＩ内蔵ＨＤＤ、一度フォーマットされているＨＤ¥n" );
    printf( "    Ｄの場合は、既に内部にシリンダ数、ヘッド数、ディフェ¥n" );
    printf( "    クト数等の情報が書き込まれていますので、Ｎを押して下¥n" );
    printf( "    さい。¥n" );
    printf( "¥n一度もフォーマットを行った事がないドライブですか？(Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        modeselectfirst( id, lun );
        formatfirst( id, lun, inl );
    }else{
        modeselect( id, lun );
        format( id, lun, inl );
    }
    printf( "物理フォーマットが終了しました。¥n" );
}


/* Mode select */
modeselect( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];
    unsigned char buf[ 20 ];
    int i;

    v[ 0 ] = C_MODSEL;
    v[ 1 ] = ( lun << 5 ) + 0x10;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 12;
    for( i = 0; i < 12; i++ ){
        buf[ i ] = 0x00;
    }
    buf[ 3 ] = 0x08;
    buf[ 10 ] = sctsiz / 256;   /* 256 or 512 or 1024 byte / sector */
/*
    buf[ 12 ] = 0x01;
    buf[ 13 ] = 0x06;
    buf[ 14 ] = 0x24;
    buf[ 15 ] = 0x08;
    buf[ 16 ] = 0x08;
    buf[ 17 ] = 0x00;
    buf[ 18 ] = 0x00;
    buf[ 19 ] = 0x00;
*/
    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}


/* Format Unit */
format( id, lun, inlv )
int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];

    v[ 0 ] = C_FMTDRV;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = inlv;
    dma = 0;    /* not DMA mode */
    printf( "¥n物理フォーマットを行っています。２～５分程度お待ち下さい。¥n" );
    if( hdcmd( id, v, sensbuf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, sensbuf );
        }else{
            printsens( sensbuf );
        }
    }
}


/* Mode select */
modeselectfirst( id, lun )
int id, lun;
{
    unsigned char  v[ 6 ], buf[ 128 ], sensbuf[ 256 ];
    int i, cyl, hd, wp, rwc;

    do{
        printf( "¥n¥nこのドライブの、シリンダ数はいくつですか？  " );
        scanf( "%d", &cyl );
        printf( "¥n                  ヘッド  数はいくつですか？  " );
        scanf( "%d", &hd );
        printf( "¥n書き込み位相補正（ライトプリコンペ）を始め¥n" );
        printf( "るシリンダはいくつですか？（不要な場合は、¥n" );
        printf( "シリンダ数と同じ数）                          " );
        scanf( "%d", &wp );
        printf( "¥n書き込み電流補正（リデュースドライトカレン¥n" );
        printf( "ト）を始めるシリンダはいくつですか？（不要¥n" );
        printf( "な場合は、シリンダ数と同じ数）                " );
        scanf( "%d", &rwc );
        printf( "¥nこれで良いですか？ (Y/N)  " );
        scanf( "%s", buf );
    }while( *buf != 'y' && *buf != 'Y' );

    for( i = 0; i < 127; i++ ){
        buf[ i ] = 0;
    }

    v[ 0 ] = C_MODSEL;
    v[ 1 ] = ( lun << 5 ) + 0x10;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 56;                /* paremeter byte no. */

    buf[  0 ] = 0x00;           /* block headder */
    buf[  1 ] = 0x00;
    buf[  2 ] = 0x00;
    buf[  3 ] = 0x08;

    buf[  4 ] = 0x00;           /* block descriptor */
    buf[  5 ] = 0x00;
    buf[  6 ] = 0x00;
    buf[  7 ] = 0x00;
    buf[  8 ] = 0x00;
    buf[  9 ] = 0x00;
    buf[ 10 ] = sctsiz / 256;   /* 256 or 512 byte / sector */
    buf[ 11 ] = 0x00;

    buf[ 12 ] = 0x04;           /* page code 4 */
    buf[ 13 ] = 0x0a;           /* parameter byte no. */
    buf[ 14 ] = 0x00;           /* No. of Cyliners */
    buf[ 15 ] = cyl >> 8;
    buf[ 16 ] = cyl & 0x00ff;
    buf[ 17 ] = hd;             /* No. of heads */
```

```
    buf[ 18 ] = 0x00;
    buf[ 19 ] = wp >> 8;        /* write precompe start cylinder */
    buf[ 20 ] = wp & 0x00ff;
    buf[ 21 ] = 0x00;           /* reduced write current start cylinder */
    buf[ 22 ] = rwc >> 8;
    buf[ 23 ] = rwc & 0x00ff;

    buf[ 24 ] = 0x03;           /* page code 3 */
    buf[ 25 ] = 0x16;
    buf[ 26 ] = 0x00;
    buf[ 27 ] = 0x00;
    buf[ 28 ] = 0x00;
    buf[ 29 ] = 0x00;
    buf[ 30 ] = 0x00;
    buf[ 31 ] = 0x00;
    buf[ 32 ] = 0x00;
    buf[ 33 ] = 0x00;
    buf[ 34 ] = 0x00;
    buf[ 35 ] = 0x00;
    buf[ 36 ] = 0x00;
    buf[ 37 ] = 0x00;
    buf[ 38 ] = 0x00;
    buf[ 39 ] = 0x00;
    buf[ 40 ] = 0x00;           /* track skew factor */
    buf[ 41 ] = 0x01;
    buf[ 42 ] = 0x00;
    buf[ 43 ] = 0x00;
    buf[ 44 ] = 0x00;
    buf[ 45 ] = 0x00;
    buf[ 46 ] = 0x00;
    buf[ 47 ] = 0x00;

    buf[ 48 ] = 0x01;   /* page code 1 */
    buf[ 49 ] = 0x06;
    buf[ 50 ] = 0x24;
    buf[ 51 ] = 0x08;
    buf[ 52 ] = 0x00;
    buf[ 53 ] = 0x00;
    buf[ 54 ] = 0x00;
    buf[ 55 ] = 0x00;

    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}

/* Format Unit */
formatfirst( id, lun, inlv )
int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];
    int dfno, i;
    struct{
        int cyl;
        int hd;
        int diff;
    }defect[ 100 ];
    char buf[ 1024 ], *ptr;

    printf( "¥nディフェクトの個数はいくつですか? " );
    scanf( "%d", &dfno );
    do{
        for( i = 0; i < dfno; i++ ){
            printf( "¥nディフェクト %3d :       シリンダは?  ", i + 1 );
            scanf( "%d", &( defect[ i ].cyl ) );
            printf( "                          ヘッド  は?  " );
            scanf( "%d", &( defect[ i ].hd ) );
            printf( "インデックスからの距離 (バイト) は?  " );
            scanf( "%d", &( defect[ i ].diff ) );
        }
        putchar( '¥n' );
        for( i = 0; i < dfno; i++ ){
            printf( "ディフェクト%4d: シリンダ%4d ヘッド%3d %6dバイト¥n",
        i + 1, defect[ i ].cyl, defect[ i ].hd, defect[ i ].diff );
        }
        printf( "¥nこれで良いですか? (Y/N)  " );
        scanf( "%s", buf );
    }while( *buf != 'y' && *buf != 'Y' );

    v[ 0 ] = C_FMTDRV;
    v[ 1 ] = ( lun << 5 ) + 0x1e;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = inlv;

    buf[ 0 ] = 0x00;
    buf[ 1 ] = 0xf1;    /* create p-list */
    buf[ 2 ] = ( dfno * 8 ) >> 8;
    buf[ 3 ] = ( dfno * 8 ) & 0x00ff;

    ptr = &( buf[ 4 ] );
    for( i = 0; i < dfno; i++ ){
        *ptr++ = defect[ i ].cyl >> 8;
        *ptr++ = defect[ i ].cyl & 0x00ff;
        *ptr++ = defect[ i ].hd;
        *ptr++ = defect[ i ].diff >>8;
        *ptr++ = defect[ i ].diff & 0x00ff;
        *ptr++ = 0;
        *ptr++ = 16;
        *ptr++ = 0;
    }
    dma = 0;    /* not DMA mode */
    printf( "¥n物理フォーマットを行っています。2～5分程度お待ち下さい。¥n" );
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}
```

〈リスト7〉SFORMMO.C

```
/* Format program for MB89352( SPC ) on J-3100
 Converted from 63C09E Assembler ( Written by Angie ) by MAL5

*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "spc.h"

unsigned scsibase;

int sctsiz;

main()
{
    unsigned char buf[ 512 ];
    int i, j, id, lun, inl, patno;
    long l, maxsct, stt;

    printf( "ＳＰＣカード用  ＭＯ－ＤｉｓｋイニシャライズユーティリティＶ１．６¥n" );
    printf( "                    Copyright by Tsuru-Zo Jul.28,1990¥n¥n" );
    printf( "  ＊＊注意＊＊  実行すると、今ＭＯに入っているデータは全て破壊されます。¥n¥n" );
    printf( "        途中で中断したい時は、Ctrl-Cを押して下さい。¥n¥n" );

    printf( "ＳＰＣカードのＩ／Ｏアドレスはどこになっていますか?  " );
    scanf( "%x", &scsibase );
```

〈リスト7〉SFORMMO.C（つづき）

```
    printf( "¥nフォーマットしたいドライブのＳＣＳＩ－ＩＤは何番ですか？ " );
    scanf( "%d", &id );
    printf( "¥nロジカルユニット番号（ドライブの番号）は何番ですか？ " );
    scanf( "%d", &lun );

    init();
    reset( id, lun );

    printf( "¥n 新品の媒体の場合、メーカであらかじめフォーマットされて¥n" );
    printf( "いない場合もありますので、一度物理フォーマットを行なって¥n" );
    printf( "ください。ただし、24分程度かかります。¥n" );
    printf( "¥n 物理フォーマットを行いますか？ (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        physformatmo( id, lun );
    }

    readcapacity( id, lun, buf );
    sctsiz = buf[ 6 ] * 256;
    maxsct = 65536L * buf[ 1 ] + 256L * buf[ 2 ] + buf[ 3 ] + 1L;

    printf( "¥n¥n¥nMOの媒体検査を行いますか？ (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        verifyunit( id, lun, maxsct );
    }
    makepart( id, lun, maxsct );

    printf( "¥nフォーマットが完了しました。¥n" );
    printf( "Config.sysにＳＰＣ８６ｍｏ／１８６ｍｏ．ＳＹＳを登録し、¥n" );
    printf( "一度リセットをして下さい。¥n" );
}

/* Format Unit */
physformatmo( id, lun, inlv )
int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];

    v[ 0 ] = C_FMTDRV;
    v[ 1 ] = lun << 5;
    v[ 2 ] = 0x03;              /* MKCDA & MKPList bit true */
    v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 0;
    dma = 0;    /* not DMA mode */
    printf( "¥n物理フォーマットを行っています。24分程度お待ち下さい。¥n" );
    if( hdcmd( id, v, sensbuf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, sensbuf );
        }else{
            printsens( sensbuf );
        }
    }
}
```

〈リスト8〉MAKEFILE（Easy Hard用）

```
scsi.sys : scsi.asm easyhard.inc
        masm scsi.asm;
        link scsi;
        exe2bin scsi scsi.sys
        del scsi.obj
        del scsi.exe

sfeh.exe : sfeh.c easyhard.h
        cl -Zp -F 4000 sfeh.c
        del sfeh.obj

sfehe.exe : sfehe.c easyhard.h
        cl -Zp -F 4000 sfehe.c
        del sfehe.obj
```

〈リスト9〉SCSI.ASM

```
;ＥＡＳＹＨＡＲＤドライバ　ｆｏｒ　ＳＣＳＩ－ＣＣＳ
;copyright by 鶴蔵
;
;V0.5   Feb.11,1990
;V0.6   Feb.19,1990                 256/512両用化
;V1.0   Feb.22,1990                 起動パラメータをサポート、一般リリース
;V1.1   Apr.02,1990                 ＰＣ／ＡＴ、３０Ｚに対応 thanks for 'そうそう'!
;V1.2   May.05,1990                 セクタサイズの検出を、ModesenseからReadCapacityに変更
;V1.3   May.06,1990                 Ｊ３１００ＳＬ／ＧＸ／ＳＧＴでも動作するように変更
;
;物理フォーマットは２５６・５１２バイト／セクタ
;
;シングルターゲット（ターゲットＩＤ、ＬＵＮはＳＣＳＩ．ＩＮＣで定義）
;
;パーティションはドライブあたり１２まで、パーティションサイズはおのおの
;最大３２ＭＢまで
;
                INCLUDE EASYHARD.INC

CODE    SEGMENT PUBLIC
                ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
;
;       デバイスヘッダー
;
;
DSKDEV  LABEL   WORD
                DD          -1              ;最終デバイス、にしておく
                DW          2000H           ;ブロックデバイス&ノンファットＩＤ
                DW          STRATEGY        ;ストラテジルーティンエントリ
                DW          DSK_INT         ;インタラプトルーティンエントリ

                PAGE
;
;       ワークエリア
;
PTRSAV  DD              0           ;リクエストヘッダアドレス退避用
TMPPHA  DB              ?           ;Phase change の検出用
RSTWT   DW              512         ;リセット送出時のウェイト時間
;
;       ＳＣＳＩコマンドバッファ（リード／ライト時用）
;
SCSI    SCSIWORK        <   >

RW_FLG          DB      ?           ;０の時リード、１の時ライト
VRFY_FLG        DB      ?           ;１の時ベリファイオン（未使用）
UNIT_NO         DB      ?           ;ユニット番号
SEC_CNT         DW      ?           ;セクタカウント
TFRMODE         DB      ?           ;転送モード(ハンドシェイクする時0,ブラインドモードで0ffh)
SSIZ            DB      ?           ;セクタサイズ(256の時1,512の時2が入る)
INITBIT         DB      ?           ;イニシエータＩＤビット
TAGTBIT         DB      ?           ;ターゲットＩＤビット
LUN             DB      ?           ;ロジカルユニットナンバ
;
;       ＤＤセクタエリア
;       ここに起動時に論理ブロック０を読み込んで来る
;
;       ここは、パーティション情報等が入る（64バイト）
;
DDSCT           DDPARAMS        <   >
;
;       ここからパーティション毎のＢＰＢが入る（１６＊１２バイト）
;
DSKBPB0         BPBTYP  <   >           ;パーティション０
DSKBPB1         BPBTYP  <   >           ;パーティション１
DSKBPB2         BPBTYP  <   >           ;パーティション２
DSKBPB3         BPBTYP  <   >           ;パーティション３
DSKBPB4         BPBTYP  <   >           ;パーティション４
DSKBPB5         BPBTYP  <   >           ;パーティション５
DSKBPB6         BPBTYP  <   >           ;パーティション６
DSKBPB7         BPBTYP  <   >           ;パーティション７
DSKBPB8         BPBTYP  <   >           ;パーティション８
DSKBPB9         BPBTYP  <   >           ;パーティション９
DSKBPB10        BPBTYP  <   >           ;パーティション１０
DSKBPB11        BPBTYP  <   >           ;パーティション１１
```

```
;
;	リクエストセンス専用のワークエリア（リトライをする関係上）
;
SENSDAT		DB	256 DUP(?)	;センスデータ格納場所
REQS		SCSIWORK	<	>

INIT_TBL:	;ＢＰＢ配列へのポインタ
		DW		DSKBPB0		;Unit #0
		DW		DSKBPB1		;Unit #1
		DW		DSKBPB2		;Unit #2
		DW		DSKBPB3		;Unit #3
		DW		DSKBPB4		;Unit #4
		DW		DSKBPB5		;Unit #5
		DW		DSKBPB6		;Unit #6
		DW		DSKBPB7		;Unit #7
		DW		DSKBPB8		;Unit #8
		DW		DSKBPB9		;Unit #9
		DW		DSKBPB10	;Unit #10
		DW		DSKBPB11	;Unit #11

CMDADDR		DW		?		;ＳＣＳＩコマンドパケットアドレス
BUFADDR		DW	2 DUP(?)	;転送先のアドレス（ＥＳ：ＤＩ）

		PAGE
;
;	各処理へのジャンプテーブル
;
JMPTBL	LABEL	WORD
		DW		DSK_INIT	; 0 Init
		DW		MEDIA_CHK	; 1 Media check
		DW		GET_BPB		; 2 Build BPB
		DW		CMD_ERR		; 3 IOCTL Input
		DW		DSK_READ	; 4 Input
		DW		EXIT		; 5 Non-Destructive input no wait
		DW		EXIT		; 6 Input status
		DW		EXIT		; 7 input flush
		DW		DSK_WRT		; 8 Output
		DW		DSK_WRT		; 9 Output with verify
		DW		EXIT		;10 Output status
		DW		EXIT		;11 Output flush
		DW		EXIT		;12 IOCTL Output
		DW		EXIT		;13 Device Open
		DW		EXIT		;14 Device Close
		DW		EXIT		;15 Removable Media
		DW		EXIT		;16 Output Until Busy
		DW		EXIT		;17 Reserved
		DW		EXIT		;18 Reserved
		DW		EXIT		;19 Generic IOCTL
		DW		EXIT		;20 Reserved
		DW		EXIT		;21 Reserved
		DW		EXIT		;22 Reserved
		DW		EXIT		;23 Get Drive Map
		DW		EXIT		;24 Set Drive Map
;
;*
;*	ストラテジルーティン
;***************
STRATP	PROC	FAR
STRATEGY:
		MOV		WORD PTR CS:[PTRSAV],BX
		MOV		WORD PTR CS:[PTRSAV+2],ES
		RET
STRATP	ENDP

		PAGE
;
;*
;*	インタラプトルーティン
;****************
DSK_INT:
		PUSH	SI	;全レジスタ,セーブ
		PUSH	AX
		PUSH	CX
		PUSH	DX
		PUSH	DI
		PUSH	BP
		PUSH	DS
		PUSH	ES
		PUSH	BX
DSK_INT1:
		LDS		BX,CS:[PTRSAV]	;リクエストヘッダアドレスを得る
		MOV		AL,[BX.UNIT]	;ユニットコード
		MOV		AH,[BX.MEDIA]	;メディアディスクリプタ
		MOV		CX,[BX.COUNT]	;セクタカウント
		MOV		DX,[BX.START]	;スタートブロック
		XCHG	AX,DI
		MOV		AL,[BX.CMD]		;コマンドコード
		CMP		AL,11
		JA		CMD_ERR			;１１以上はエラーにしている（手抜き）
		XOR		AH,AH
		SHL		AX,1
		MOV		SI,OFFSET JMPTBL
		ADD		SI,AX
		XCHG	AX,DI
		LES		DI,[BX.TRANS]	;転送先アドレスを得る
		PUSH	CS
		POP		DS				;ＤＳ←ＣＳ
		JMP		[SI]			;各処理に分岐する
	PAGE
;*
;*	終了ルーティン
;****************
CMD_ERR:						;コマンドエラーの場合
		MOV		AL,3			;ＡＬに、エラーコード３をセット
ERR_EXIT:
		MOV		AH,10000001B	;エラービットとＤｏｎｅビットをセット
		JMP		SHORT EXIT1

EXITP	PROC	FAR
EXIT:							;ノーエラー
		MOV		AH,00000001B	;Ｄｏｎｅビットのみセット
EXIT1:
		LDS		BX,CS:[PTRSAV]	;リクエストヘッダアドレスを得る
		MOV		[BX.STATUS],AX	;ステータスバイトをセット

		POP		BX				;レジスタ復帰
		POP		ES
		POP		DS
		POP		BP
		POP		DI
		POP		DX
		POP		CX
		POP		AX
		POP		SI
		RET
EXITP	ENDP
	PAGE
;*****************
;*	コマンド処理ルーティン
;*****************

;*
;*	コマンド１：メディアチェック
;*****************
MEDIA_CHK:						;メディアが交換されたかのチェック
		LDS		BX,[PTRSAV]
		MOV		BYTE PTR [BX.TRANS],1	;常に「交換されてない」を返す
		JMP		EXIT


;*
;*	コマンド２：ビルドＢＰＢ
;*******************
GET_BPB:						;ＢＰＢを返す
		MOV		AH,0
		SHL		AX,1
		LEA		DI,INIT_TBL
		ADD		DI,AX
		MOV		BX,[DI]
		MOV		AH,BYTE PTR [BX]+10	;ＦＡＴ－ＩＤバイト
```

〈リスト9〉SCSI.ASM（つづき）

```
        MOV     SI,BX                   ;ＢＰＢのアドレス
        LDS     BX,[PTRSAV]             ;リクエストヘッダのアドレスを得る
        MOV     [BX.MEDIA],AH           ;メディアディスクリプタ
        MOV     [BX.COUNT],SI           ;ＢＰＢのアドレスをセット
        MOV     [BX.COUNT+2],CS
        JMP     EXIT

;*
;*  コマンド4：インプット
;*****************
DSK_READ:                               ;ディスクリードルーティン
        MOV     RW_FLG,0                ;「リードモード」をセット
        MOV     VRFY_FLG,0
        JMP     RW_COMMON

;*
;*  コマンド8：アウトプット
;*****************
DSK_WRT:                                ;ディスクライトルーティン
        MOV     RW_FLG,1                ;「ライトモード」をセット
        MOV     VRFY_FLG,0              ;ベリファイはＯＦＦ
        JMP     RW_COMMON

;*
;*  コマンド9：アウトプット&ベリファイ
;*********************
DSK_WRTV:                               ;ライト&ベリファイ
        MOV     RW_FLG,1                ;「ライトモード」をセット
        MOV     VRFY_FLG,1              ;ベリファイはＯＮ
        JMP     RW_COMMON               ;と言っても、現在ベリファイは
                                        ;サポートしてない。
;
;   リード／ライトの共通ルーティン
;
RW_COMMON:
        JCXZ    EXIT                    ;セクタカウントが０の時は何もしない
RWC1:   MOV     UNIT_NO,AL              ;ユニットナンバ、セクタ数を
        MOV     SEC_CNT,CX              ;セーブ

        CMP     RW_FLG,0                ;リード？
        JNE     WRT
                                        ;リードルーティン
        MOV     SCSI.CMDS,C$REDSEC      ;ＳＣＳＩのリードコマンドをセット
        JMP     RW10
WRT:
        MOV     SCSI.CMDS,C$WRTSEC      ;ＳＣＳＩのライトコマンドをセット
;
RW10:
        MOV     AH,0                    ;パーティション分のセクタ
        SHL     AX,1                    ;オフセットの計算
        SHL     AX,1
        LEA     BX,DDSCT.SCTOFS         ;ＤＤセクタに書かれていた値と
        ADD     BX,AX                   ;リクエストヘッダの値を加算する

        ADD     DX,WORD PTR [BX]        ;セクタオフセットを加算
        MOV     AL,0
        ADC     AL,BYTE PTR [BX+2]

        MOV     AH,BYTE PTR SSIZ
        CMP     AH,01H                  ;セクタサイズ＝２５６の場合は
        JNE     RW20
        SHL     CX,1                    ;セクタカウント＊＝２
        SHL     DX,1                    ;セクタアドレス＊＝２
        RCL     AL,1
RW20:
        OR      AL,BYTE PTR LUN
        MOV     SCSI.SECH,AL            ;ＳＣＳＩコマンドパケットの作成
        MOV     SCSI.SECM,DH
        MOV     SCSI.SECL,DL
        MOV     SCSI.BLOCK,CL
        MOV     SCSI.CNTL,0

        LEA     AX,SCSI                 ;ＡＸにコマンドパケットアドレス
```

```
        MOV     TFRMODE,0FFH            ;転送モードはブラインド転送
        CALL    HDCMD                   ;ＳＣＳＩハンドルルーティンを呼ぶ

        JC      RWERR                   ;セレクションタイムアウト
        MOV     AL,SCSI.STAT
        CMP     AL,0                    ;ノーエラーならそのままリターン
        JE      RWRTN

        CALL    REQSENS                 ;チェックの場合はリクエストセンスをして
        JC      RWERR
        MOV     AL,SENSDAT+2
        CMP     AL,1                    ;リカバードエラーは無視する
        JE      RWRTN
        CMP     AL,6                    ;リセットコンディションの時はリードライト
        JE      RWRETRY                 ;出来てないからリトライ

        CLC
        JMP     RWERR                   ;それ以外は本当のエラー

RWRETRY:                                ;リトライする
        LEA     AX,SCSI
        MOV     TFRMODE,0FFH
        CALL    HDCMD

        JC      RWERR
        MOV     AL,SCSI.STAT
        CMP     AL,0
        JNE     RWERR                   ;リトライはしない

RWRTN:                                  ;正常終了
        LDS     BX,[PTRSAV]             ;リクエストヘッダアドレスを得る
        MOV     [BX.COUNT],CX           ;転送されたセクタ数をセット

        JMP     EXIT

RWERR:                                  ;異常終了
        JNC     RWE1
        MOV     AL,E$NOTRDY             ;ＡＬにエラーコードをセットして
        JMP     RWERET
RWE1:
        TEST    RW_FLG,01H
        JNE     RWE2
        MOV     AL,E$READ
        JMP     RWERET
RWE2:
        MOV     AL,E$WRITE
RWERET:
        JMP     ERR_EXIT                ;ＥＲＲ＿ＥＸＩＴにジャンプ
        RET
;
;
;       リクエストセンス
;
REQSENS:
        PUSH    DI                      ;転送アドレスは保存しておく
        PUSH    ES

        MOV     REQS.CMDS,C$REQSEN      ;専用バッファアドレスにコマンド
        MOV     AL,BYTE PTR LUN
        MOV     REQS.SECH,AL            ;パケットをセット
        MOV     REQS.SECM,00H           ;この後でコマンドリトライをする
        MOV     REQS.SECL,00H           ;事があるのでバッファを分けてある
        MOV     REQS.BLOCK,128
        MOV     REQS.CNTL,00H

        LEA     AX,REQS
        LEA     DI,SENSDAT              ;ここにセンスデータが格納される

        PUSH    CS
        POP     ES                      ;CS -> ES

        MOV     TFRMODE,0               ;ソフト転送モードを選択
        CALL    HDCMD

        POP     ES
        POP     DI
```

〈リスト9〉SCSI.ASM（つづき）

```
		RET
	PAGE
;
;	ＳＣＳＩハンドリングルーティン
;	ＡＸにコマンドパケットアドレス
;	ＥＳ：ＤＩに転送バッファアドレス
;
HDCMD:
		MOV		CMDADDR,AX		;アドレスのセーブ
		MOV		BUFADDR,DI
		MOV		BUFADDR+2,ES
		PUSH	CS
		POP		ES

		GETP	P$CNT			;コントロールレジスタを読む
		TEST	AL,T$RST		;リセットビットを調べる
		JE		HDCMD0			;セットしてなければレジュームはされていない
		CALL	SCSI_INIT		;一度レジュームされているので、ＳＣＳＩバス
						;をリセットする
HDCMD0:
		MOV		CX,1000
HDCMD1:
		GETP	P$SNS
		TEST	AL,T$BSY		;Busyがリセットされていれば、
		JNE		HDCMD2			;多分ＨＤＤはつながってるんだろう
		LOOP	HDCMD1
		JMP		EREX0
HDCMD2:						;セレクションフェーズ
		MOV		AL,BYTE PTR INITBIT
		OR		AL,BYTE PTR TAGTBIT	;Bus-IDの送出
		NOT		AL
		MOV		DX,P$DATA
		OUT		DX,AL
		PUTPI	P$CNT,T$ATN+T$SEL		;Selection bit true

		MOV		CX,4000
HDCMD11:
		GETP	P$SNS
		TEST	AL,T$BSY		;Busyがセットされていれば、
		JE		HDCMDCMD		;Selection成功
		LOOP	HDCMD11

EREX0:	MOV		AL,E$NOTRDY		;セレクションタイムアウト
		PUSH	AX
		PUSH	CX
		PUTPI	P$CNT,T$ATN+T$RST	;バスリセットを送出
		MOV		CX,100
		LOOP	$
		PUTPI	P$CNT,T$ATN		;バスリセット解除
		MOV		CX,10000
		LOOP	$
		POP		CX
		POP		AX
		JMP		HDCMDERR

HDCMDCMD:					;コマンドフェーズ
		PUTPI	P$DATA,0ffh		;データラインクリア
		PUTPI	P$CNT,T$ATN		;selection False
HDCMDC1:
		GETP	P$SNS			;ＲＥＱがtrueになるまで待つ
		TEST	AL,T$REQ
		JNE		HDCMDC1

		TEST	AL,T$CD			;コマンドフェーズかチェック
		JNE		HDCMDC1

		MOV		DI,CMDADDR		;ＤＩ←コマンドパケットアドレス

		CALL	MANTFR			;マニュアル転送する
		JNC		HDCMDTFR

		JMP		HDCMDERR		;エラーの場合

HDCMDTFR:					;トランスファーフェーズ
		GETP	P$SNS			;ＲＥＱがtrueになるまで待つ
		TEST	AL,T$REQ
		JNE		HDCMDTFR

		TEST	AL,T$MSG+T$CD+T$IO	;ステータスフェーズの場合もあるので
		JE		HDCMDSTS		;チェックする

		MOV		DI,BUFADDR		;ＤＩ←データバッファアドレス
		MOV		ES,BUFADDR+2		;ＥＳ←データバッファセグメント

		TEST	TFRMODE,0FFH		;ブラインド転送モード？
		JE		HDMAN

		CALL	BLINDTFR		;ブラインド転送
		JMP		HDCT1
HDMAN:
		CALL	MANTFR			;マニュアル転送
HDCT1:
		PUSH	CS
		POP		ES
HDCMDSTS:					;ステータスフェーズ
		GETP	P$SNS
		TEST	AL,T$REQ		;ＲＥＱが１になるまで待つ
		JNE		HDCMDSTS

		TEST	AL,T$MSG+T$CD+T$IO
		JNE		HDCMDSTS

		MOV		BX,CMDADDR
		LEA		DI,[BX].STAT
		CALL	MANTFR			;ステータスバイトを得る
HDCMDMES:					;メッセージインフェーズ
		GETP	P$SNS
		TEST	AL,T$REQ		;ＲＥＱが１になるまで待つ
		JNE		HDCMDMES

		AND		AL,T$MSG+T$CD
		CMP		AL,T$MSG
		JNE		HDCMDMES

		PUSH	CS
		POP		ES

		MOV		BX,CMDADDR
		LEA		DI,[BX].MESS
		CALL	MANTFR			;メッセージバイトを得る
HDCMDBF:					;バスフリーフェーズ
		GETP	P$SNS
		TEST	AL,T$BSY		;ビジーが０になるまで待つ
		JE		HDCMDBF

HDCMDBF1:					;ディスコネクト
		MOV		DI,BUFADDR		;バッファアドレスの復帰
		MOV		ES,BUFADDR+2
		CLC					;正常終了
		RET

HDCMDERR:					;異常終了
		MOV		DI,BUFADDR
		MOV		ES,BUFADDR+2
		STC
		RET
	PAGE
;
;	ブラインド転送ルーティン（ハンドシェイクを行わない）
;	本当は、やりたくないのだが、このハードではこうでもしないと
;	実用的なスピードが得られないのでまあしょうがない。
;	現在は、データリード／ライトコマンド時のみ使用
;	データバッファアドレスはＥＳ：ＤＩ
;
BLINDTFR:
		PUSH	DI
		PUSH	SI
		MOV		BX,CMDADDR		;ＳＣＳＩリクエストヘッダから
		MOV		AH,0
```

〈リスト9〉SCSI.ASM（つづき）

```
        MOV     AL,[BX].BLOCK           ;転送すべきバイト数／５１２を得て
        MOV     CX,AX                   ;でそれをＣＸレジスタにセットして

        MOV     AL,BYTE PTR SSIZ        ;セクタサイズが５１２の場合は
        CMP     AL,01H
        JE      BLITFR05                ;ループカウントを２倍にする
        SHL     CX,1
BLITFR05:
        MOV     BX,P$DATA
        PUSH    DS
        PUSH    ES
        POP     DS                      ;DS←ES
        CLD

        PUTPI   P$DATA,0ffh             ;Data Lineをクリア
BLITFR10:
        GETP    P$SNS
        TEST    AL,T$REQ                ;ＲＥＱがtrueになるまで待つ
        JNE     BLITFR10

BLITFR15:                               ;データイン／アウトをチェックする
        TEST    AL,T$IO
        JNE     BLIOUT0

BLIIN0:
        PUTPI   P$CNT,T$IOS+T$IODIR+T$ATN       ;Data portを入力モードに設定
BLIIN1:
        PUSH    CX
        MOV     CX,256                  ;256bytesずつblind転送
BLIIN2:
        GETP    P$SNS
        TEST    AL,T$REQ                ;ＲＥＱがtrueになるまで待つ
        JNE     BLIIN2

        TEST    AL,T$CD                 ;フェーズが変わっていればエラー
        JE      BLITFRERR

        MOV     DX,P$SNS
BLIIN4:
        IN      AL,DX
        TEST    AL,T$REQ                ;ＲＥＱがtrueになるまで待つ
        JNE     BLIIN4

BLIIN5:
        DEC     DX                      ;DX←P$DATA
        IN      AL,DX                   ;データを得る
        NOT     AL
        STOSB
        MOV     AX,(T$IOS+T$IODIR+T$ATN)*256+T$IOS+T$IODIR+T$ATN+T$ACK
        MOV     DX,P$CNT
        OUT     DX,AL                   ;Set ACKビット

        MOV     DX,P$CNT
        MOV     AL,AH
        OUT     DX,AL                   ;Reset ACKビット

        MOV     DX,P$SNS
        LOOP    BLIIN4

        POP     CX
        LOOP    BLIIN1

        PUTPI   P$CNT,T$ATN     ;Data portを出力モードに設定

        JMP     BLITFR20

BLIOUT0:
        MOV     SI,DI
BLIOUT1:
        PUSH    CX
        MOV     CX,511
        MOV     DX,P$SNS
BLIOUT2:
        IN      AL,DX
        TEST    AL,T$REQ                ;ＲＥＱがtrueになるまで待つ
        JNE     BLIOUT2

        TEST    AL,T$CD                 ;フェーズが変わっていればエラー
        JE      BLITFRERR0

        LODSB                           ;最初の１バイトだけハンドシェイクする
        NOT     AL
        PUTP    P$DATA

        PUTPI   P$CNT,T$ATN+T$ACK       ;Set ACKビット
        MOV     DX,P$SNS
BLIOUT3:                                ;REQビットがfalseになるまで待つ
        IN      AL,DX
        TEST    AL,T$REQ
        JE      BLIOUT3

        PUTPI   P$CNT,T$ATN             ;Reset ACKビット

        MOV     DX,P$SNS
BLIOUT4:
        IN      AL,DX
        TEST    AL,T$REQ                ;ＲＥＱがtrueになるまで待つ
        JNE     BLIOUT4

BLIOUT5:                                ;データアウトの場合
        LODSB
        NOT     AL
        DEC     DX                      ;DX←P$DATA
        OUT     DX,AL                   ;データを送出
        MOV     AX,(T$ATN)*256+T$ATN+T$ACK
        MOV     DX,P$CNT
        OUT     DX,AL                   ;Set ACKビット
        MOV     AL,AH
        OUT     DX,AL                   ;Reset ACKビット

        MOV     DX,P$SNS
        LOOP    BLIOUT4

        POP     CX
        LOOP    BLIOUT1

BLITFR20:
        POP     DS
        JMP     BLITFREXIT

BLITFRERR:
        POP     CX
        PUTPI   P$DATA,0ffh             ;Data Lineをクリア
        POP     SI
        POP     DI                      ;異常終了
        STC
        RET
BLITFREXIT:
        PUTPI   P$DATA,0ffh             ;Data Lineをクリア
        POP     SI
        POP     DI                      ;正常終了
        CLC
        RET
BLITFRERR0:
        POP     CX
        PUTPI   P$DATA,0ffh             ;Data Lineをクリア
        POP     DS
        JMP     BLITFREXIT
    PAGE
;
;       マニュアル転送ルーティン
;       フェーズが変化するまで転送する
;       バッファアドレスはＥＳ：ＤＩ
;
MANTFR:
        PUSH    DI
        PUTPI   P$DATA,0ffh             ;Data Lineをクリア

MANTFR10:
        GETP    P$SNS
        TEST    AL,T$REQ                ;ＲＥＱがtrueになるまで待つ
```

```
        JNE     MANTFR10

        AND     AL,T$MSG+T$CD+T$IO      ;現在のバスフェーズをストア
        MOV     TMPPHA,AL
MANTFR20:
        TEST    AL,T$IO                 ;InかOutかをチェック
        JNE     MANOUT
MANIN:                                  ;DATA IN フェーズ
        PUTPI   P$CNT,T$IOS+T$IODIR+T$ATN ;Data portを入力モードに設定

        MOV     CX,100                  ;Data setup delay
        LOOP    $

        GETP    P$DATA                  ;Dataを得る
        NOT     AL
        MOV     ES:[DI],AL

        PUTPI   P$CNT,T$ATN     ;Data portを出力モードに設定

        JMP     SETACK
MANOUT:                                 ;DATA OUTフェーズ
        MOV     AL,ES:[DI]
        NOT     AL
        PUTP    P$DATA                  ;Dataを送出
        MOV     CX,100                  ;Data setup delay
        LOOP    $

SETACK:                                 ;Set ACKビット
        PUTPI   P$CNT,T$ATN+T$ACK
WAITREQF:                               ;REQビットがfalseになるまで待つ
        GETP    P$SNS
        TEST    AL,T$REQ
        JE      WAITREQF

        PUTPI   P$CNT,T$ATN     ;Reset ACKビット

        INC     DI                      ;バッファのポインタを進める
MANTFR30:                               ;次のREQが来るまで待つ
        GETP    P$SNS
        TEST    AL,T$REQ
        JE      MANTFR40
        TEST    AL,T$BSY                ;Bus Free Phaseなら終了
        JNE     MANTFREXIT
        JMP     MANTFR30
MANTFR40:
        AND     AL,T$MSG+T$CD+T$IO      ;今度のフェーズを得る
        CMP     AL,TMPPHA               ;今までと同じ？
        JE      MANTFR20                ;ならもう1バイト
MANTFREXIT:
        PUTPI   P$DATA,0ffh             ;Data Lineをクリア
        POP     DI                      ;終了
        CLC
        RET

;*ＳＣＳＩバスのリセット信号をクリアする
;*通常は、Ｉｎｉｔルーティンより一回呼ばれるだけだが、Ｊ３１００ＳＬ／ＳＳ
;*の場合はレジュームされるたびにＣａｌｌされる。
;******************************
SCSI_INIT:
        PUSH    AX
        PUSH    CX
        PUSH    DX
        PUTPI   P$CNT,T$ATN+T$RST       ;リセット信号をセット
        MOV     CX,100
        LOOP    $
        PUTPI   P$CNT,T$ATN             ;リセット信号をクリア

        MOV     CX,WORD PTR RSTWT       ;指定秒数待つ
SCI1:
        PUSH    CX
        MOV     CX,5500
        LOOP    $
        POP     CX
        LOOP    SCI1
        POP     DX
        POP     CX
        POP     AX
        RET
;*
;*  イニットルーティン（デバイス登録時に1度だけ呼ばれる）
;*  改版してちょっと長くなったが、起動後は切り放されるから良しとしよう
;************************************
DSK_INIT:                               ;まずＳＣＳＩバスのイニシャライズ
        LEA     DX,MSG                  ;Ver.,etc.を表示
        MOV     AH,09H
        INT     21H

                                        ;ＳＣＳＩアドレスの初期値を設定
        MOV     INITBIT,I$INITBIT       ;イニシエータＩＤ＝７
        MOV     TAGTBIT,I$TAGTBIT       ;デフォルトのターゲットＩＤ＝０
        MOV     LUN,I$LUN               ;デフォルトのロジカルユニットＮｏ．＝０

        LDS     BX,[PTRSAV]             ;リクエストヘッダアドレスを得る

        MOV     DI,WORD PTR [BX.COUNT]
        MOV     ES,WORD PTR [BX.COUNT+2]
        MOV     AL,BYTE PTR [BX.COUNT+4] ;最初のドライブ番号を得る
        PUSH    CS
        POP     DS
        MOV     BYTE PTR STTDRV,AL

        CALL    PRMCHK                  ;Config.sysのparameterをチェック
DINIT0:                                 ;リードキャパシティ実行
        MOV     SCSI.CMDS,C$RDCAP
        MOV     AL,BYTE PTR LUN
        MOV     SCSI.SECH,AL
        MOV     SCSI.SECM,00H
        MOV     SCSI.SECL,00H
        MOV     SCSI.BLOCK,00H
        MOV     SCSI.CNTL,00H
        MOV     SCSI.EXT1,00H
        MOV     SCSI.EXT2,00H
        MOV     SCSI.EXT3,00H           ;PMI bit false
        MOV     SCSI.EXT4,00H           ;flag and link bit false

        LEA     AX,SCSI
        PUSH    CS
        POP     ES                      ;ＥＳ←ＣＳ
        LEA     DI,DDSCT                ;ＤＩ←ＤＤＳＣＴ

        MOV     TFRMODE,00H             ;マニュアル転送を選択
        CALL    HDCMD
        JNC     DINIT01
        JMP     DIERR
DINIT01:
        CMP     SCSI.STAT,00H           ;ステータスチェック
        JE      DINIOEND                ;ノーエラー

        CALL    REQSENS                 ;センスデータを得る
        JNC     DINIT02
        JMP     DIERR
DINIT02:
        MOV     AL,BYTE PTR SENSDAT+2
        CMP     AL,01H                  ;リカバードエラーは無視する
        JE      DINIOEND
        CMP     AL,06H                  ;リセットコンディションだともう1度
        JE      DINIT0
DINIOEND:
        MOV     AL,BYTE PTR DDSCT+6     ;セクタサイズを得る
        MOV     BYTE PTR SSIZ,AL

DINIT1:                                 ;ＤＤセクタの読み込み
        MOV     SCSI.CMDS,C$REDSEC
        MOV     AL,BYTE PTR LUN
        MOV     SCSI.SECH,AL            ;ブロックアドレス０から
        MOV     SCSI.SECM,00H
```

```
        MOV     SCSI.SECL,00H
        MOV     SCSI.BLOCK,01H          ;１セクタだけ読み込む
        MOV     SCSI.CNTL,00H

        LEA     AX,SCSI
        PUSH    CS
        POP     ES                      ;E'S←ＣＳ
        LEA     DI,DDSCT

        MOV     TFRMODE,00H             ;マニュアル転送を選択
        CALL    HDCMD
        JC      DIERR
        CMP     SCSI.STAT,00H           ;ステータスチェック
        JE      DIEX1                   ;ノーエラー

        CALL    REQSENS                 ;センスデータを得る
        JC      DIERR
        MOV     AL,BYTE PTR SENSDAT+2
        CMP     AL,01H                  ;リカバードエラーは無視する
        JE      DIEX1
        CMP     AL,06H                  ;リセットコンディションだともう１度
        JE      DINIT1
DIERR:
        LEA     DX,MSGERR               ;失敗のメッセージを表示
        MOV     AH,09H
        INT     21H

        PUSH    DS
        MOV     AL,0
        LDS     BX,[PTRSAV]             ;リクエストヘッダアドレスを得る
        MOV     [BX.MEDIA],AL           ;ドライブ数←０

        MOV     WORD PTR [BX.TRANS],OFFSET DSK_INIT
        MOV     WORD PTR [BX.TRANS+2],CS

        MOV     WORD PTR [BX.COUNT],OFFSET INIT_TBL
        MOV     WORD PTR [BX.COUNT+2],CS
                                        ;ＢＰＢ配列のポインタ
        POP     DS

        STC
        MOV     AL,E$NOTRDY             ;エラーコードのセット
        JMP     ERR_EXIT
DIEX1:
        LEA     DX,MSGCMP               ;成功のメッセージを表示
        MOV     AH,09H
        INT     21H

        MOV     CH,0
        MOV     CL,BYTE PTR DDSCT.DRVNUM ;ＤＤセクタから、パーティションの
        MOV     DL,BYTE PTR STTDRV      ;数を得る
        ADD     DL,'A'
        LEA     BX,DSKBPB0+9
DIEX2:
        PUSH    DX
        MOV     AH,02H                  ;ドライブ番号の表示
        INT     21H
        MOV     DL,':'
        INT     21H
        MOV     AL,BYTE PTR [BX]
        SHR     AL,1
        ADD     AL,02H                  ;１ＭＢ以下を四捨五入
        SHR     AL,1
        AND     AL,07EH
        MOV     AH,0                    ;AX = Drive_Capacity(MB) * 2
        PUSH    BX
        LEA     BX,CAPTABLE
        ADD     BX,AX
        MOV     DL,BYTE PTR [BX]
        MOV     AH,02H
        INT     21H
        MOV     DL,BYTE PTR [BX+1]
        INT     21H
        POP     BX
        LEA     DX,MSGCAP
        MOV     AH,09H
        INT     21H
        ADD     BX,16                   ;ＢＰＢアドレスを進める
        POP     DX
        INC     DL                      ;ドライブ番号＋＝１
        LOOP    DIEX2

        LEA     DX,MSGEND
        MOV     AH,09H
        INT     21H

        MOV     AL,BYTE PTR DDSCT.DRVNUM
        PUSH    DS
        LDS     BX,[PTRSAV]             ;リクエストヘッダアドレスを得る
        MOV     [BX.MEDIA],AL           ;ドライブ数←パーティションの数

        MOV     WORD PTR [BX.TRANS],OFFSET DSK_INIT
        MOV     WORD PTR [BX.TRANS+2],CS

        MOV     WORD PTR [BX.COUNT],OFFSET INIT_TBL
        MOV     WORD PTR [BX.COUNT+2],CS
                                        ;ＢＰＢ配列のポインタ
        POP     DS
        CLC                             ;エラー無し

        JMP     EXIT

;
;       起動パラメータのチェック
;       文字列のポインタはＥＳ：ＤＩ
;
PRMCHK:                                 ;search '/' char.
        CALL    PRMGET1
        JC      PRMCHKEND               ;if End of line
        CMP     AL,'/'
        JE      PRMCHK1
        JMP     PRMCHK                  ;次の文字
;
PRMCHK1:                                ;'/'の次の文字を得る
        CALL    PRMGET1
        JC      PRMCHKERR
        AND     AL,0DFH                 ;toupper(*s)
        CMP     AL,'I'
        JE      PRMCHKID                ;'/I'の場合
        CMP     AL,'L'
        JE      PRMCHKLUN               ;'/L'の場合
        CMP     AL,'W'
        JE      PRMCHKWT                ;'/W'の場合
        JMP     PRMCHKERR

PRMCHKID:                               ;change target ID
        CALL    PRMGET1
        JC      PRMCHKERR
        SUB     AL,'0'
        JL      PRMCHKERR
        CMP     AL,6
        JA      PRMCHKERR
        MOV     AH,0
        MOV     CX,AX
        MOV     AL,01H
        CMP     CX,0
        JE      PRMID2
PRMID1: SHL     AL,1
        LOOP    PRMID1
PRMID2:
        MOV     BYTE PTR TAGTBIT,AL
        JMP     PRMCHK
;
PRMCHKLUN:                              ;change LUN
        CALL    PRMGET1
        JC      PRMCHKERR
        SUB     AL,'0'
        JL      PRMCHKERR
        CMP     AL,3
        JA      PRMCHKERR
        MOV     CX,5
```

〈リスト9〉 SCSI.ASM（つづき）

```
PRMLUN1:                                        ;LUN = unit no. * 20H
        SHL     AL,1
        LOOP    PRMLUN1

        MOV     CS:BYTE PTR LUN,AL
        JMP     PRMCHK

PRMCHKWT:                               ;change wait time of Reset condition
        CALL    PRMGET1
        JC      PRMCHKERR
        SUB     AL,'0'
        JL      PRMCHKERR
        CMP     AL,1
        JL      PRMCHKERR
        CMP     AL,9
        JA      PRMCHKERR
        MOV     AH,AL
        MOV     AL,0
        MOV     CS:WORD PTR RSTWT,AX
        JMP     PRMCHK
;
PRMCHKERR:
        LEA     DX,MSGPRMERR            ;パラメータ不正のメッセージを表示
        MOV     AH,09H
        INT     21H

PRMCHKEND:
        RET
PRMGET1:
        MOV     AL,ES:[DI]                      ;search '/'
        INC     DI
        CMP     AL,0DH
        JE      PRMGETEND
        CMP     AL,0AH
        JE      PRMGETEND
        CLC
        RET
PRMGETEND:
        STC
        RET
;
;   起動時用のワークエリア
;
STTDRV  DB      ?

;
;   起動メッセージ
;
MSG     DB      0AH,0DH
        DB      'Easy-Hard SCSI Driver V1.3 (256 or 512 Bytes/Sct)'
        DB      '  for J3100 / PC/AT',0AH,0DH
        DB      '  Copyright by Tsuru-Zoh, May.06,1990',0AH,0DH
        DB      '   SCSI-BUS Reset executing. Wait a moment,please.'
MSGEND  DB      0AH,0DH,'$'
MSGCMP  DB      '  Initialize Complete! Drive ','.','$'
MSGERR  DB      '  Initialize Error.',0AH,0DH,'$'
MSGPRMERR:
        DB      '    Invalid parameter.',0AH,0DH,'$'
MSGCAP  DB      'MB ','.','$'
CAPTABLE:
        DB      ' 0 1 2 3 4 5 6 7 8 9101112131415'
        DB      '16171819202122232425262728293031 32'
;
CODE    ENDS
        END
```

〈リスト10〉 EASYHARD.INC

```
;
;   I／Oアドレスの定義
;
P$DATA      EQU     378H    ;データレジスタ（双方向）
P$SNS       EQU     379H    ;センスレジスタ（入力）
P$CNT       EQU     37AH    ;コントロールレジスタ（出力）

I$INITID    EQU     7               ;イニシエータID＝7
I$INITBIT   EQU     80H

I$TAGTID    EQU     0               ;デフォルトのターゲットID＝0
I$TAGTBIT   EQU     01H

I$LUN       EQU     00H             ;デフォルトのロジカルユニット番号＝0

;
;   SCSIのコマンド定義
;
C$TSTRDY    EQU     00H
C$REZERO    EQU     01H
C$REQSEN    EQU     03H
C$FMTDRV    EQU     04H
C$REDSEC    EQU     08H
C$WRTSEC    EQU     0AH
C$SEEK      EQU     0BH
C$MODSEL    EQU     15H
C$MODSEN    EQU     1AH
C$RDCAP     EQU     25H
;
;   リクエストヘッダ構造体の定義
;
REQ_HEAD    STRUC
CMDLEN      DB      ?               ;コマンドのバイト数
UNIT        DB      ?               ;ユニット番号
CMD             DB      ?           ;コマンド番号
STATUS      DW      ?               ;実行ステータス格納場所
                DB  8 DUP(?)        ;システムリザーブ
MEDIA       DB      ?               ;メディアディスクリプタ
TRANS       DD      ?               ;転送バッファアドレス
COUNT       DW      ?               ;転送セクタ数
START       DW      ?               ;開始ブロック番号
REQ_HEAD    ENDS

;
;   SCSIコマンドパケット構造体の定義
;
SCSIWORK    STRUC
CMDS        DB      ?               ;SCSIのコマンドコード
SECH        DB      ?               ;セクタ番号
SECM        DB      ?               ;
SECL        DB      ?               ;
BLOCK       DB      ?               ;転送セクタ数
CNTL        DB      ?               ;
EXT1        DB      ?               ;グループ1コマンド用拡張
EXT2        DB      ?               ;
EXT3        DB      ?               ;
EXT4        DB      ?               ;
STAT        DB      ?               ;実行ステータス格納場所
MESS        DB      ?               ;実行メッセージ格納場所
SCSIWORK    ENDS

;
;   DDセクタ領域のバイト定義
;
DDPARAMS    STRUC
DRVNUM      DB      ?               ;このドライブのパーティションの数
DDRSV1      DB      ?
SCTOFS      DD      12 DUP (?)      ;パーティションの開始するセクタ番号
DDRSV2      DB      14 DUP(?)       ;リザーブ
DDPARAMS    ENDS

;
;   MS－DOSのBPB構造体の定義
;   1ドライブあたり16バイト確保している
;
BPBTYP      STRUC
SCTSIZ      DW      ?               ;セクタサイズ
ALLSIZ      DB      ?               ;アロケーションユニットあたりのセクタ数
RSVSCT      DW      ?               ;リザーブセクタ数
```

〈リスト10〉 EASYHARD.INC（つづき）

```
FATCPY      DB     ?          ;ＦＡＴのコピー数
DIRENT      DW     ?          ;ルートディレクトリのエントリ数
SCTVOL      DW     ?          ;このユニットの総セクタ数
MEDIAD      DB     ?          ;メディアディスクリプタバイト
FATNO       DW     ?          ;ＦＡＴのセクタ数
BPBRSV      DB     3 DUP(?)
BPBTYP      ENDS

;
;    エラーコードの定義
;
E$WP        EQU    00H        ;WRITE PROTECT
E$ERRUNIT   EQU    01H        ;ILLEGAL UNIT
E$NOTRDY    EQU    02H        ;NOT READY
E$ERRCMD    EQU    03H        ;ILLEGAL UNIT
E$CRC       EQU    04H        ;CRC ERROR
E$REQLEN    EQU    05H        ;ERROR @ REQ.LENGTH
E$SEEK      EQU    06H        ;SEEK ERROR
E$MEDIA     EQU    07H        ;MEDIA ERROR
E$SECTOR    EQU    08H        ;SECTOR NOT FOUND
E$PAPER     EQU    09H        ;PAPER OUT
E$WRITE     EQU    0AH        ;WRITE ERROR
E$READ      EQU    0BH        ;READ ERROR
E$ERR       EQU    0CH        ;GLOBAL ERROR
E$CHG       EQU    0FH        ;MEDIA CHANGE ERROR

;
;    バスのビット定義
;
;    センスレジスタ
;
T$MSG   EQU        80H
T$REQ   EQU        40H
T$BSY   EQU        20H
T$CD    EQU        10H
T$IO    EQU        08H
;
;    コントロールレジスタ
;
T$IOS   EQU        80H
T$IODIR EQU        20H
T$RST   EQU        08H
T$ATN   EQU        04H
T$SEL   EQU        02H
T$ACK   EQU        01H
;
;    レジスタ構造体の定義
REGSTER STRUC
AHm         DB          ?
ALm         DB          ?
BHm         DB          ?
BLm         DB          ?
CHm         DB          ?
CLm         DB          ?
DXm         DW          ?
DIm         DW          ?
DSm         DW          ?
REGSTER ENDS
```

```
;
;    Ｉ／Ｏポートアクセスのマクロ定義
;
CLRP    MACRO   ADDR
                MOV         DX, ADDR
                IN          AL, DX
                MOV         AL, 0
                OUT         DX, AL
                ENDM

GETP    MACRO   ADDR
                MOV         DX, ADDR
                IN          AL, DX
                ENDM

PUTP    MACRO   ADDR
                MOV         DX, ADDR
                OUT         DX, AL
                ENDM

PUTPI   MACRO   ADDR, DATA
                MOV         AL, DATA
                MOV         DX, ADDR
                OUT         DX, AL
                ENDM

PUTPW   MACRO   ADDR
                MOV         DX, ADDR
                OUT         DX, AL
                MOV         AL, AH
                OUT         DX, AL
                ENDM

GETPW   MACRO   ADDR
                PUSH    CX
                MOV         DX, ADDR
                IN          AL, DX
                MOV         CL, AL
                IN          AL, DX
                MOV         AH, AL
                MOV         AL, CL
                POP         CX
                ENDM
```

〈リスト11〉 SFEH.C

```
/*
  Format program for EasyHard and SCSI-CCS HDD
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "easyhard.h"

int sctsiz;

main()
{
    unsigned char buf[ 512 ];
    int i, j, id, lun, inl, patno;
    long l, maxsct, stt;

    printf( "ＥａｓｙＨａｒｄ用　ＨＤＤイニシャライズユーティリティＶ１．３¥n" );
    printf( "                        Copyright by 鶴蔵 May.06,1990¥n¥n" );
    printf( "   ＊＊注意＊＊　実行すると、今ＨＤＤに入っているデータは全て破壊されます。¥n¥n" );
    printf( "     途中で中断したい時は、Ctrl-Cを押して下さい。¥n¥n" );

    printf( "¥nフォーマットしたいドライブのＳＣＳＩ－ＩＤは何番ですか？ " );
    scanf( "%d", &id );
    printf( "¥nロジカルユニット番号（ドライブの番号）は何番ですか？ " );
    scanf( "%d", &lun );

    reset( id, lun );

    printf( "¥n¥n   このドライバで使用する為には、　２５６又は５１２バ¥n" );
    printf( "    イト／セクタに物理フォーマットされている必要がありま¥n" );
    printf( "    す。¥n" );
    printf( "     ２５６又は５１２バイト／セクタでフォーマットされて¥n" );
    printf( "    いない場合、インタリーブ値を変更する場合は物理フォー¥n" );
    printf( "    マットを行って下さい。¥n" );
    printf( "¥n 物理フォーマットを行いますか？ (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        physformat( id, lun );
    }

    readcapacity( id, lun, buf );
    if( ( buf[ 6 ] != 1 ) && ( buf[ 6 ] != 2 ) ){
        printf( "¥n¥n＊＊＊物理セクタサイズが２５６、５１２バイト以外になっています。¥n" );
        printf( "一度、物理フォーマットを行って下さい。¥n" );
        exit( 1 );
    }
    if( sctsiz == 0 ){
        sctsiz = buf[ 6 ] * 256;
    }
    if( ( buf[ 6 ] * 256 ) != sctsiz ){
        printf( "¥n¥n＊＊＊＊選択されたセクタサイズと物理セクタサイズが異なっています。¥n" );
        printf( "選択されたセクタサイズ：%d   物理セクタサイズ：%d¥n",
            sctsiz, buf[ 10 ] * 256 );
        sctsiz = buf[ 10 ] * 256;
        printf( "無視して処理を続行しますか？ (Y/N)" );
        scanf( "%s", buf );
        if( ( *buf != 'y' ) && ( *buf != 'Y' ) ){
            exit( 1 );
        }
    }
    maxsct = 65536L * buf[ 1 ] + 256L * buf[ 2 ] + buf[ 3 ] + 1L;

    printf( "¥n¥n¥nハードディスクの媒体検査を行いますか？ (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        verifyunit( id, lun, maxsct );
    }
    makepart( id, lun, maxsct );

    printf( "¥nフォーマットが完了しました。¥n" );
    printf( "ＣＯＮＦＩＧ．ＳＹＳにＳＣＳＩ．ＳＹＳを登録し、リセットをして下さい。¥n" );
}
```

```
/* SCSI BUS-Reset */
reset( id, lun )
int id, lun;
{
    long l, t;

    printf( "¥nＳＣＳＩバスリセットを実行中です。１０秒お待ち下さい。¥n" );
    outp( P_CNT, ATN + RST );
    for( l = 0L; l < 300L; l++ ){
    }
    outp( P_CNT, ATN );
    for( l = 0L; l < 100000L; l++ ){
    }
    l = time( (long*)0 );
    while( ( t = ( time( (long*)0 ) - l ) ) < 10L ){
        printf( "%ld¥r", t );
    }
    printf( "        ¥n" );
}


/* physical format */
physformat( id, lun )
int id,lun;
{
    int inl;
    char buf[ 32 ];

    printf( "¥n¥n    ＯＳ９と共用する場合は、２５６バイト／セクタとすると¥n" );
    printf( "    便利です。それ以外の時は５１２バイト／セクタにした方¥n" );
    printf( "    が容量が増える場合が多いです。また、５１２バイト／セ¥n" );
    printf( "    クタしか出来ないドライブもあります。¥n¥n" );
    do{
        printf( "セクタサイズを幾つにしますか？（２５６・５１２）  " );
        scanf( "%d", &sctsiz );
    }while( ( sctsiz != 256 ) && ( sctsiz != 512 ) );

    printf( "¥n¥n    コントローラにトラックバッファが有る場合は、必ずイ ¥n" );
    printf( "    ンタリーブファクタを１にして下さい。¥n" );
    printf( "    トラックバッファが無い場合は、J3100SS では８程度が¥n" );
    printf( "    適当な筈ですが、コントローラ、システムによって異なり¥n" );
    printf( "    ますので各自実験してみて下さい。¥n" );
    printf( "¥nインタリーブファクタは、いくつにしますか？ " );
    scanf( "%d", &inl );

    printf( "¥n¥n    Ｙを押すと、ＨＤＤのシリンダ、ヘッド数、ディフェク ¥n" );
    printf( "    ト情報入力画面になります。¥n" );
    printf( "    ＳＣＳＩ内蔵ＨＤＤ、一度フォーマットされているＨＤ¥n" );
    printf( "    Ｄの場合は、既に内部にシリンダ数、ヘッド数、ディフェ¥n" );
    printf( "    クト数等の情報が書き込まれていますので、Ｎを押して下¥n" );
    printf( "    さい。¥n" );
    printf( "¥n一度もフォーマットを行った事がないドライブですか？(Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        modeselectfirst( id, lun );
        formatfirst( id, lun, inl );
    }else{
        modeselect( id, lun );
        format( id, lun, inl );
    }
    printf( "物理フォーマットが終了しました。¥n" );
}


verifyunit( id, lun, maxblock )
int id, lun;
long maxblock;
{
    long l, lt, le, badblock;
    unsigned char buf[ 256 ];
    int times, i;

    printf( "¥n¥n最大ブロック番号は、%ldです。¥n", maxblock );
    printf( "不良ブロックがある場合は、自動的に交替処理を行います。¥n" );
    printf( "¥n媒体検査を何回繰り返しますか？  " );
    scanf( "%d", &times );
    putchar( '¥n' );
    for( i = 0; i < times; i++ ){
        printf( "媒体検査  :  %d 回目¥n", i + 1 );
        for( l = 0L; l < maxblock; l += 1024L ){
            printf( "ブロック %ld¥r", l );
            lt = l;
            le = ( ( lt + 1024L ) > maxblock )? maxblock - l: 1024L;
            while( verify( id, lun, lt, le, buf ) ){
                badblock = 65536L * buf[ 4 ] + 256L * buf[ 5 ] + buf[ 6 ];
                reassign( id, lun, badblock );
                le -= ( badblock - lt + 1 );
                lt = badblock + 1L;
                if( lt == ( l + 1024L ) ){
                    break;
                }
            }
        }
    }
    printf( "  媒体検査が終了しました。  ¥n" );
}

makepart( id, lun, maxsct )
int id, lun;
long maxsct;
{
static struct{
    int pn;
    long sctofs[ 20 ];
}ddsct;

static struct{
    int         ssiz;
    char        allsiz;
    int         rsvsct;
    char        fatcpy;
    int         dirent;
    unsigned    sctvol;
    char        mediad;
    int         fatno;
}bpb[ 20 ] = {
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
    { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }
};

int patno, i, j;
unsigned vol;
long restsct, l;
char buf[ 512 ];

maxsct = maxsct / ( ( sctsiz == 256 ) ? 2L: 1L ) - 3L;
do{
    restsct = maxsct;
    printf( "¥nディスクの総セクタ数は、%ld(%ldバイト)です。¥n",
          maxsct, maxsct * 512L );
    printf( "¥nパーティションを幾つにしますか？ （１～１２）  " );
    scanf( "%d", &patno );
    for( i = 0; i < patno; i++ ){
        do{
            printf( "¥n残りセクタ数は、%ld(%ldバイト)です。¥n", restsct, restsct * 512L );
            printf( "パーティション %d に何セクタ割当てますか？(最大%ld)",
             i + 1, (restsct > 65535L )? 65535L: restsct );
            scanf( "%d", &vol );
        }while( vol > 65535 || vol > restsct );
        bpb[ i ].sctvol = vol;
        restsct -= vol;
        if( vol > 20480 ){
            bpb[ i ].allsiz = 4;
        }else if( vol > 10240 ){
            bpb[ i ].allsiz = 2;
```

〈リスト11〉 SFEH.C （つづき）

```
        }else{
            bpb[ i ].allsiz = 1;
        }
        bpb[ i ].fatno = vol / bpb[ i ].allsiz * 2 / 512 + 1;
    }
    putchar( '¥n' );
    for( i = 0; i < patno; i++ ){
        printf( "パーティション %3d: %uセクタ(%ldバイト)¥n", i + 1, bpb[ i ].sctvol, 512L * bpb[ i ].sctvol );
    }
    printf( "残りセクタ：%ld¥n", restsct );
    printf( "これで良いですか？ (Y/N)  " );
    scanf( "%s", buf );
}while( *buf != 'y' && *buf != 'Y' );

printf( "¥nＦＡＴとディレクトリの初期化を行っています。¥n" );
restsct = 3L;
for( i = 0; i < patno; i++ ){
    printf( "パーティション %3d ¥n", i + 1 );
    for( j = 0; j < 512; j++ ){
        buf[ j ] = 0;
    }
    for( l = 1; l < 2L * bpb[ i ].fatno + 33L; l++ ){
        swrite( id, lun, restsct + l, buf );
    }
    buf[ 0 ] = 0xfe;
    buf[ 1 ] = 0xff;
    buf[ 2 ] = 0xff;
    swrite( id, lun, restsct + 1L, buf );
    swrite( id, lun, restsct + 1L + bpb[ i ].fatno, buf );
    memcpy( &( buf[ 3 ] ), "EHD V1.2", 8 );
    memcpy( &( buf[ 11 ] ), &( bpb[ i ] ), sizeof( bpb[ i ] ) );
    swrite( id, lun, restsct, buf );
    restsct += bpb[ i ].sctvol;
}

restsct = 3L;
ddsct.pn = patno;
for( i = 0; i < 12; i++ ){
        ddsct.sctofs[ i ] = restsct;
        restsct += bpb[ i ].sctvol;
    }
    memcpy( &( buf[ 0 ] ), &ddsct, sizeof( ddsct ) );
    for( i = 0; i < 12; i++ ){
        memcpy( &( buf[ 64 + i * 16 ] ), &( bpb[ i ] ), sizeof( bpb[ i ] ) );
    }
    swrite( id, lun, 0L, buf );

}

/* Verify */
verify( id, lun, start, range, buf )
int id, lun;
long start, range;
unsigned char *buf;
{
    unsigned char v[ 10 ], sensbuff[ 128 ];
    int sens;

    v[ 0 ] = 0x2f;
    v[ 1 ] = lun << 5;
    v[ 6 ] = v[ 9 ] = 0x00;

    v[ 2 ] = ( start >> 24 ) & 0x000000ffL;
    v[ 3 ] = ( start >> 16 ) & 0x000000ffL;
    v[ 4 ] = ( start >> 8 ) & 0x000000ffL;
    v[ 5 ] = start & 0x000000ffL;
    v[ 7 ] = ( range >> 8 ) & 0x000000ffL;
    v[ 8 ] = range & 0x000000ffL;

    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        sens = requestsense( id, lun, buf );
        if( sens == 1 || sens == 3 ){
            return( 1 );
        }else{
            printsens( buf );
```

```
        }
    }
    return( 0 );
}

/* Reassign block */
reassign( id, lun, badblock )
int id, lun;
long badblock;
{
    unsigned char v[ 6 ], buf[ 128 ];

    printf( "ブロック%ld が不良です。交替処理を行います。¥n", badblock );

    v[ 0 ] = 0x07;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    buf[ 0 ] = buf[ 1 ] = buf[ 2 ] = 0x00;
    buf[ 3 ] = 4;
    buf[ 4 ] = ( badblock >> 24 ) & 0x000000ffL;
    buf[ 5 ] = ( badblock >> 16 ) & 0x000000ffL;
    buf[ 6 ] = ( badblock >> 8 ) & 0x000000ffL;
    buf[ 7 ] = badblock & 0x000000ffL;

    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, buf ) ){
            printsens( buf );
        }
    }
}

/* Test Unit ready */
testready( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], buf[ 80 ];
    int i;

    v[ 0 ] = C_TSTRDY;
    v[ 1 ] = ( lun << 5 );
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;

    dma = 0;    /* not DMA mode */

    return ( hdcmd( id, v, buf ) );
}

/* Mode select */
modeselect( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];
    unsigned char buf[ 20 ];
    int i;

    v[ 0 ] = C_MODSEL;
    v[ 1 ] = ( lun << 5 ) + 0x10;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 12;
    for( i = 0; i < 12; i++ ){
        buf[ i ] = 0x00;
    }
    buf[ 3 ] = 0x08;
    buf[ 10 ] = ( ( sctsiz == 256 ) ? 0x01: 0x02 ); /* 256 or 512 byte / sector */
/*
    buf[ 12 ] = 0x01;
    buf[ 13 ] = 0x06;
    buf[ 14 ] = 0x24;
    buf[ 15 ] = 0x08;
    buf[ 16 ] = 0x08;
    buf[ 17 ] = 0x00;
    buf[ 18 ] = 0x00;
    buf[ 19 ] = 0x00;
```

```
*/
    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}

/* read capacity */
readcapacity( id, lun, buf )
int id, lun;
unsigned char *buf;
{
    unsigned char  v[ 10 ], sensbuf[ 256 ];

    v[ 0 ] = C_RDCAP;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;   /*logical block address is 0 */
    v[ 6 ] = v[ 7 ] = 0x00;
    v[ 8 ] = 0x00;              /* PMI bit false */
    v[ 9 ] = 0x00;              /* flag and link false */

    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}

/* Format Unit */
format( id, lun, inlv )
int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];

    v[ 0 ] = C_FMTDRV;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = inlv;
    dma = 0;    /* not DMA mode */
    printf( "¥n物理フォーマットを行っています。2～5分程度お待ち下さい。¥n" );
    if( hdcmd( id, v, sensbuf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, sensbuf );
        }else{
            printsens( sensbuf );
        }
    }
}


/* Mode select */
modeselectfirst( id, lun )
int id, lun;
{
unsigned char  v[ 6 ], buf[ 128 ], sensbuf[ 256 ];
int i, cyl, hd, wp, rwc;

do{
    printf( "¥n¥nこのドライブの、シリンダ数はいくつですか？  " );
    scanf( "%d", &cyl );
    printf( "¥n                 ヘッド 数はいくつですか？  " );
    scanf( "%d", &hd );
    printf( "¥n書き込み位相補正（ライトプリコンペ）を始め¥n" );
    printf( "るシリンダはいくつですか？（不要な場合は、¥n" );
    printf( "シリンダ数と同じ数)                         " );
    scanf( "%d", &wp );
    printf( "¥n書き込み電流補正（リデュースドライトカレン¥n" );
    printf( "ト）を始めるシリンダはいくつですか？（不要¥n" );
    printf( "な場合は、シリンダ数と同じ数)               " );
    scanf( "%d", &rwc );
    printf( "¥nこれで良いですか？ (Y/N)  " );
    scanf( "%s", buf );
}while( *buf != 'y' && *buf != 'Y' );

for( i = 0; i < 127; i++ ){
    buf[ i ] = 0;
}

v[ 0 ] = C_MODSEL;
v[ 1 ] = ( lun << 5 ) + 0x10;
v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
v[ 4 ] = 56;                    /* paremeter byte no. */

buf[  0 ] = 0x00;               /* block headder */
buf[  1 ] = 0x00;
buf[  2 ] = 0x00;
buf[  3 ] = 0x08;

buf[  4 ] = 0x00;               /* block descriptor */
buf[  5 ] = 0x00;
buf[  6 ] = 0x00;
buf[  7 ] = 0x00;
buf[  8 ] = 0x00;
buf[  9 ] = 0x00;
buf[ 10 ] = ( ( sctsiz == 256 ) ? 0x01: 0x02 ); /* 256 or 512 byte / sector */
buf[ 11 ] = 0x00;

buf[ 12 ] = 0x04;               /* page code 4 */
buf[ 13 ] = 0x0a;               /* parameter byte no. */
buf[ 14 ] = 0x00;               /* No. of Cyliners */
buf[ 15 ] = cyl >> 8;
buf[ 16 ] = cyl & 0x00ff;
buf[ 17 ] = hd;                 /* No. of heads */
buf[ 18 ] = 0x00;
buf[ 19 ] = wp >> 8;            /* write precompe start cylinder */
buf[ 20 ] = wp & 0x00ff;
buf[ 21 ] = 0x00;               /* reduced write current start cylinder */
buf[ 22 ] = rwc >> 8;
buf[ 23 ] = rwc & 0x00ff;

buf[ 24 ] = 0x03;               /* page code 3 */
buf[ 25 ] = 0x16;
buf[ 26 ] = 0x00;
buf[ 27 ] = 0x00;
buf[ 28 ] = 0x00;
buf[ 29 ] = 0x00;
buf[ 30 ] = 0x00;
buf[ 31 ] = 0x00;
buf[ 32 ] = 0x00;
buf[ 33 ] = 0x00;
buf[ 34 ] = 0x00;
buf[ 35 ] = 0x00;
buf[ 36 ] = 0x00;
buf[ 37 ] = 0x00;
buf[ 38 ] = 0x00;
buf[ 39 ] = 0x00;
buf[ 40 ] = 0x00;               /* track skew factor */
buf[ 41 ] = 0x01;
buf[ 42 ] = 0x00;
buf[ 43 ] = 0x00;
buf[ 44 ] = 0x00;
buf[ 45 ] = 0x00;
buf[ 46 ] = 0x00;
buf[ 47 ] = 0x00;

buf[ 48 ] = 0x01;   /* page code 1 */
buf[ 49 ] = 0x06;
buf[ 50 ] = 0x24;
buf[ 51 ] = 0x08;
buf[ 52 ] = 0x00;
buf[ 53 ] = 0x00;
buf[ 54 ] = 0x00;
buf[ 55 ] = 0x00;
```

〈リスト11〉SFEH.C（つづき）

```
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}

/* Format Unit */
formatfirst( id, lun, inlv )
int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];
    int dfno, i;
    struct{
        int cyl;
        int hd;
        int diff;
    }defect[ 100 ];
    char buf[ 1024 ], *ptr;

    printf( "¥nディフェクトの個数はいくつですか? " );
    scanf( "%d", &dfno );
    do{
        for( i = 0; i < dfno; i++ ){
            printf( "¥nディフェクト %3d :   シリンダは?  ", i + 1 );
            scanf( "%d", &( defect[ i ].cyl ) );
            printf( "              ヘッド  は?  " );
            scanf( "%d", &( defect[ i ].hd ) );
            printf( "インデックスからの距離 (バイト) は?  " );
            scanf( "%d", &( defect[ i ].diff ) );
        }
        putchar( '¥n' );
        for( i = 0; i < dfno; i++ ){
            printf( "ディフェクト%4d: シリンダ%4d ヘッド%3d %6dバイト¥n",
        i + 1, defect[ i ].cyl, defect[ i ].hd, defect[ i ].diff );
        }
        printf( "¥nこれで良いですか? (Y/N)  " );
        scanf( "%s", buf );
    }while( *buf != 'y' && *buf != 'Y' );

    v[ 0 ] = C_FMTDRV;
    v[ 1 ] = ( lun << 5 ) + 0x1e;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = inlv;

    buf[ 0 ] = 0x00;
    buf[ 1 ] = 0xf1;    /* create p-list */
    buf[ 2 ] = ( dfno * 8 ) >> 8;
    buf[ 3 ] = ( dfno * 8 ) & 0x00ff;

    ptr = &( buf[ 4 ] );
    for( i = 0; i < dfno; i++ ){
        *ptr++ = defect[ i ].cyl >> 8;
        *ptr++ = defect[ i ].cyl & 0x00ff;
        *ptr++ = defect[ i ].hd;
        *ptr++ = defect[ i ].diff >>8;
        *ptr++ = defect[ i ].diff & 0x00ff;
        *ptr++ = 0;
        *ptr++ = 16;
        *ptr++ = 0;
    }
    dma = 0;    /* not DMA mode */
    printf( "¥n物理フォーマットを行っています。2～5分程度お待ち下さい。¥n" );
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}
```

```
/* Write 1 Sector */
swrite( id, lun, lsn, buf )
int id, lun;
long lsn;   /* logical sector number */
unsigned char *buf; /* data buffer */
{
    unsigned char v[ 6 ];
    unsigned char sensbuf[ 256 ];

    if( sctsiz == 256 ){
        lsn *= 2L;
    }
    v[ 0 ] = C_WRTSEC;
    v[ 4 ] = ( ( sctsiz == 256 )? 2: 1 );
    v[ 5 ] = 0x00;
    v[ 1 ] = ( lun << 5 ) + ( lsn >> 16 ) & 0x00ff;
    v[ 2 ] = ( lsn >> 8 ) & 0x00ff;
    v[ 3 ] = lsn & 0x00ff;
    dma = 0;    /* write 256 bytes */
    scsimode = 0x0b88;
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}

requestsense( id, lun, buf )
int id, lun;
unsigned char *buf;
{
    /* check error. do request sense. */
    unsigned char v[ 6 ];

    v[ 0 ] = C_REQSEN;
    v[ 1 ] = lun << 5;   /* set Lun */
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 255;
    dma = 0x00;
    hdcmd( id, v, buf );
    return( buf[ 2 ] );
}

printsens( buf )
unsigned char *buf;
{
    printf( "Sense key = %02X, Extend sense code = %02X¥n", buf[ 2 ], buf[ 12 ] );
    dump( buf );
    exit( 99 );
}

/* output command to HDD */
hdcmd( id, sc, buf )
int id;
unsigned char *sc, *buf;
{
    int treg;   /* temp. reg. */
    int idb;

    idb = 0x0001 << id;

    /* selection phase */
/*  printf( "Selection.¥n" );*/
    outp( P_DATA, (INITBIT+idb) ); /* set target & init. ID */
    outp( P_CNT, ATN+SEL );              /* selection bit true */

    for( treg = 0; treg < 1000; treg ++ ){
     /* wait responding time ( 200us max. ) */
    }
```

```
    if( inp( P_SNS ) & BSY ){   /* if BSY is false */
        printf( " couldn't select device.¥n " );
        exit( 99 );
    }
    outp( P_DATA, 0xff );       /* clear data line */
    outp( P_CNT, ATN );     /* selection complete. reset selection bit. */
    /* command phase */
/*  printf( "Command.¥n" );*/
    do{
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait while REQ = false */
        }
        treg &= CD;
    }while( treg ); /* wait until command phase */
    mantfr( sc );   /* output command data */

    /* data transfer phase */
/*  printf( "Data Transfer.¥n" );*/
    while( ( treg = inp( P_SNS ) ) & REQ ){
        /* wait while REQ = false */
    }
    if( ( treg & ( MSG + CD + IO ) ) != 0 ){/* status phase? (MSG=f,CD,IO=t) */
        /* no, data phase. */
        mantfr( buf );
    }

    /* status phase */
/*  printf( "Status.¥n" );*/
    do{
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait while REQ = false */
        }
    }while( ( treg & ( MSG + CD + IO ) ) != 0 ); /* wait until MSG:f CD,IO:t */

    tfrsub( &sttbuf );  /* get Status byte */

    /* message phase */
/*  printf( "Message.¥n" );*/
    do{
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait until REQ = true */
        }
        treg &= ( MSG + CD );
    }while( treg != MSG );
    tfrsub( &msgin );   /* get message byte */

    /* bus free phase */
/*  printf( "Bus free.¥n" );*/
    while( !( inp( P_SNS ) & BSY ) ){
        /* wait until BUSY = false */
    }
    /* disconnect */
    treg = sttbuf;
    if( treg == 0x01 ){
        /* parity error */
        printf( "parity error occured.¥n" );
        exit( 99 );
    }else if( treg == 0x02 ){
        /* check condition */
        return( 1 );
    }else{
        return( treg );
    }
}

/* manual Data transfer routine */
mantfr( buf )
unsigned char *buf;
{
    int treg;   /* temporary register */
    int tphase; /* initial phase storage */

    while( ( treg = inp( P_SNS ) ) & REQ ){
        /* wait until REQ = true */
    }
    tphase = treg & ( MSG + CD + IO );
    do{
        tfrsub( buf++ );
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait until REQ = true */
        }
        treg &= ( MSG + CD + IO );
    }while( treg == tphase );       /* repeat until phase change */

    return( 0 );

}

/* sub routine: 1 byte transfer */
tfrsub( buf )
unsigned char *buf;
{
    int treg;   /* temp. reg. */

    outp( P_DATA,0xff );                /* clear data line */
    while( ( treg = inp( P_SNS ) ) & REQ ){
        /* wait while REQ = false */
    }
    if( ( treg & IO ) == 0 ){
        outp( P_CNT, IOS+DIR+ATN );
        /* I/O IN */
        *buf = ~inp( P_DATA );
        outp( P_CNT, ATN );
    }else{
        /* I/O OUT */
        outp( P_DATA, ~(*buf) );
    }
    outp( P_CNT, ATN + ACK );           /* set ACK bit */
    while( !( inp( P_SNS ) & REQ ) ){
        /* wait until REQ = false */
    }
    outp( P_CNT, ATN );                 /* reset ACK bit */
    outp( P_DATA, 0xff );               /* clear data line */
}


dump( buf )
unsigned char *buf;
{
    int i, j;
    unsigned d;

    printf( "      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF¥n" );

    for( i = 0; i < 256; i += 16 ){
        printf( "%04X: ", i );
        for( j = 0; j < 16; j++ ){
            printf( "%02X ", buf[ i + j ] );
        }
        for( j = 0; j < 16; j++ ){
            d = buf[ i + j ];
            if( ( d >= ' ' ) && ( d < 0x7f ) ){
                putchar( d );
            }else{
                putchar( '.' );
            }
        }
        putchar( '¥n' );
    }
}
```

〈リスト12〉 SFEHE.C

```
/*
    Format program for EasyHard and SCSI-CCS HDD
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "easyhard.h"

int sctsiz;

main()
{
    unsigned char buf[ 512 ];
    int i, j, id, lun, inl, patno;
    long l, maxsct, stt;

    printf( "HDD initialize utility V1.3 for EasyHard and SCSI-CCS HDD¥n" );
    printf( "                   Copyright by Tsuru-Zoh May.06,1990¥n" );
    printf( "¥n  *** Warning! *** all data on HDD will be lost!¥n¥n" );
    printf( "                  press [Ctrl-C] to exit.¥n¥n" );

    printf( "Enter the  SCSI-ID to format (0 to 6): " );
    scanf( "%d", &id );
    printf( "Enter the SCSI-LUN to format (0 to 3): " );
    scanf( "%d", &lun );

    reset( id, lun );

    printf( "¥n  Physical format desired ? (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        physformat( id, lun );
    }

    readcapacity( id, lun, buf );
    if( ( buf[ 6 ] != 1 ) && ( buf[ 6 ] != 2 ) ){
        printf( "Physical sector size must be 256 or 512.¥n" );
        printf( "Do physical format.¥n" );
        exit( 1 );
    }
    if( sctsiz == 0 ){
        sctsiz = buf[ 6 ] * 256;
    }
    if( ( buf[ 6 ] * 256 ) != sctsiz ){
        printf( "Selected Sector size : %d   Physical Sector size : %¥n",
            sctsiz, buf[ 10 ] * 256 );
        sctsiz = buf[ 10 ] * 256;
        printf( "Ignore and continue ? (Y/N)" );
        scanf( "%s", buf );
        if( ( *buf != 'y' ) && ( *buf != 'Y' ) ){
            exit( 1 );
        }
    }
    maxsct = 65536L * buf[ 1 ] + 256L * buf[ 2 ] + buf[ 3 ] + 1L;
    printf( "¥nMedia Verify desired ? (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        verifyunit( id, lun, maxsct );
    }
    makepart( id, lun, maxsct );
    printf( "¥nFormat complete.¥n" );
    printf( "Add 'SCSI.SYS' to config.sys, and reboot please.¥n" );
}

/* SCSI BUS-Reset */
reset( id, lun )
int id, lun;
{
    long l, t;

    printf( "¥nSCSI BUS-Reset executing. Wait 10 sec.¥n" );
    outp( P_CNT, ATN + RST );
    for( l = 0L; l < 300L; l++ ){
    }
    outp( P_CNT, ATN );
    for( l = 0L; l < 100000L; l++ ){
    }
    l = time( (long*)0 );
    while( ( t = ( time( (long*)0 ) - l ) ) < 10L ){
        printf( "%ld¥r", t );
    }
    printf( "        ¥n" );
}
/* physical format */
physformat( id, lun )
int id,lun;
{
    int inl;
    char buf[ 32 ];

    do{
        printf( "Enter the sector size  ? (256 or 512) " );
        scanf( "%d", &sctsiz );
    }while( ( sctsiz != 256 ) && ( sctsiz != 512 ) );

    printf( "Enter the Interleave factor ? " );
    scanf( "%d", &inl );

    printf( "Is this the first time to format this HDD ? (Y/N) " );
    scanf( "%s", buf );
    if( *buf == 'y' || *buf == 'Y' ){
        modeselectfirst( id, lun );
        formatfirst( id, lun, inl );
    }else{
        modeselect( id, lun );
        format( id, lun, inl );
    }
    printf( "¥nPhysical format complete.¥n" );
}

verifyunit( id, lun, maxblock )
int id, lun;
long maxblock;
{
    long l, lt, le, badblock;
    unsigned char buf[ 256 ];
    int times, i;

    printf( "¥nNo. of Sector is %ld.¥n", maxblock );
    printf( "How many times do you want to verify the Media ?" );
    scanf( "%d", &times );
    putchar( '¥n' );
    for( i = 0; i < times; i++ ){
        printf( "Media Verify %d times¥n", i + 1 );
        for( l = 0L; l < maxblock; l += 1024L ){
            printf( "Block  %ld¥r", l );
            lt = l;
            le = ( ( lt + 1024L ) > maxblock )? maxblock - 1: 1024L;
            while( verify( id, lun, lt, le, buf ) ){
                badblock = 65536L * buf[ 4 ] + 256L * buf[ 5 ] + buf[ 6 ];
                reassign( id, lun, badblock );
                le -= ( badblock - lt + 1 );
                lt = badblock + 1L;
                if( lt == ( l + 1024L ) ){
                    break;
                }
            }
        }
    }
    printf( "¥nMedia Verify complete.¥n" );
}

makepart( id, lun, maxsct )
int id, lun;
long maxsct;
{
    static struct{
        int pn;
        long sctofs[ 20 ];
    }ddsct;
    static struct{
```

```
	int		ssiz;
	char		allsiz;
	int		rsvsct;
	char		fatcpy;
	int		dirent;
	unsigned	sctvol;
	char		mediad;
	int		fatno;
}bpb[ 20 ] = {
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 },
	{ 512, 4, 1, 2, 512, 40000, 0xf8, 40 }, { 512, 4, 1, 2, 512, 40000, 0xf8, 40 }
};
int patno, i, j;
unsigned vol;
long restsct, l;
char buf[ 512 ];
maxsct = maxsct / ( ( sctsiz == 256 ) ? 2L: 1L ) - 3L;
do{
	restsct = maxsct;
	printf( "YnWhole sector No. is %ld(%ld Bytes).Yn",
		maxsct, maxsct * 512L );
	printf( "How many partitions do you want ? ( 1 to 12 ) " );
	scanf( "%d", &patno );
	for( i = 0; i < patno; i++ ){
		do{
			printf( "YnRest sector No. is %ld(%ld Bytes).Yn", restsct, restsct * 512L );
			printf( "How many sectors do you assigns to partition %d ?(Max.%ld) ",
 i + 1, (restsct > 65535L )? 65535L: restsct );
			scanf( "%d", &vol );
		}while( vol > 65535 || vol > restsct );
		bpb[ i ].sctvol = vol;
		restsct -= vol;
		if( vol > 20480 ){
			bpb[ i ].allsiz = 4;
		}else if( vol > 10240 ){
			bpb[ i ].allsiz = 2;
		}else{
			bpb[ i ].allsiz = 1;
		}
		bpb[ i ].fatno = vol / bpb[ i ].allsiz * 2 / 512 + 1;
	}
	putchar( 'Yn' );
	for( i = 0; i < patno; i++ ){
		printf( "Partition %3d: %uSectors(%ldBytes)Yn", i + 1, bpb[ i ].sctvol, 512L * bpb[ i ].sctvol );
	}
	printf( "Rest sectors %ldYn", restsct );
	printf( "Are you sure ? (Y/N)  " );
	scanf( "%s", buf );
}while( *buf != 'y' && *buf != 'Y' );

printf( "YnInitializing FAT and directories.Yn" );
restsct = 3L;
for( i = 0; i < patno; i++ ){
	printf( "Partition  %3d Yn", i + 1 );
	for( j = 0; j < 512; j++ ){
		buf[ j ] = 0;
	}

	for( l = 1; l < 2L * bpb[ i ].fatno + 33L; l++ ){
		swrite( id, lun, restsct + l, buf );
	}
	buf[ 0 ] = 0xfe;
	buf[ 1 ] = 0xff;
	buf[ 2 ] = 0xff;
	swrite( id, lun, restsct + 1L, buf );
	swrite( id, lun, restsct + 1L + bpb[ i ].fatno, buf );
	memcpy( &( buf[ 3 ] ), "EHD V1.2", 8 );
	memcpy( &( buf[ 11 ] ), &( bpb[ i ] ), sizeof( bpb[ i ] ) );
	swrite( id, lun, restsct, buf );
	restsct += bpb[ i ].sctvol;
}

	restsct = 3L;
	ddsct.pn = patno;
	for( i = 0; i < 12; i++ ){
		ddsct.sctofs[ i ] = restsct;
		restsct += bpb[ i ].sctvol;
	}
	memcpy( &( buf[ 0 ] ), &ddsct, sizeof( ddsct ) );
	for( i = 0; i < 12; i++ ){
		memcpy( &( buf[ 64 + i * 16 ] ), &( bpb[ i ] ), sizeof( bpb[ i ] ) );
	}
	swrite( id, lun, 0L, buf );

}

/* Verify */
verify( id, lun, start, range, buf )
int id, lun;
long start, range;
unsigned char *buf;
{
	unsigned char v[ 10 ], sensbuff[ 128 ];
	int sens;

	v[ 0 ] = 0x2f;
	v[ 1 ] = lun << 5;
	v[ 6 ] = v[ 9 ] = 0x00;

	v[ 2 ] = ( start >> 24 ) & 0x000000ffL;
	v[ 3 ] = ( start >> 16 ) & 0x000000ffL;
	v[ 4 ] = ( start >> 8 ) & 0x000000ffL;
	v[ 5 ] = start & 0x000000ffL;
	v[ 7 ] = ( range >> 8 ) & 0x000000ffL;
	v[ 8 ] = range & 0x000000ffL;

	dma = 0;	/* not DMA mode */
	if( hdcmd( id, v, buf ) ){
		sens = requestsense( id, lun, buf );
		if( sens == 1 || sens == 3 ){
			return( 1 );
		}else{
			printsens( buf );
		}
	}
	return( 0 );
}


/* Reassign block */
reassign( id, lun, badblock )
int id, lun;
long badblock;
{
	unsigned char v[ 6 ], buf[ 128 ];

	printf( "Block %ld is Bad. Reassign it.Yn", badblock );

	v[ 0 ] = 0x07;
	v[ 1 ] = lun << 5;
	v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
	buf[ 0 ] = buf[ 1 ] = buf[ 2 ] = 0x00;
	buf[ 3 ] = 4;
	buf[ 4 ] = ( badblock >> 24 ) & 0x000000ffL;
	buf[ 5 ] = ( badblock >> 16 ) & 0x000000ffL;
	buf[ 6 ] = ( badblock >> 8 ) & 0x000000ffL;
	buf[ 7 ] = badblock & 0x000000ffL;

	dma = 0;	/* not DMA mode */
	if( hdcmd( id, v, buf ) ){
		if( requestsense( id, lun, buf ) ){
			printsens( buf );
		}
	}
}

/* Test Unit ready */
testready( id, lun )
```

〈リスト12〉SFEHE.C（つづき）

```
int id, lun;
{
    unsigned char v[ 6 ], buf[ 80 ];
    int i;

    v[ 0 ] = C_TSTRDY;
    v[ 1 ] = ( lun << 5 );
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;

    dma = 0;    /* not DMA mode */

    return ( hdcmd( id, v, buf ) );
}

/* Mode select */
modeselect( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];
    unsigned char buf[ 20 ];
    int i;

    v[ 0 ] = C_MODSEL;
    v[ 1 ] = ( lun << 5 ) + 0x10;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 12;
    for( i = 0; i < 12; i++ ){
        buf[ i ] = 0x00;
    }
    buf[ 3 ] = 0x08;
    buf[ 10 ] = ( ( sctsiz == 256 ) ? 0x01: 0x02 ); /* 256 or 512 byte / sector */
/*
    buf[ 12 ] = 0x01;
    buf[ 13 ] = 0x06;
    buf[ 14 ] = 0x24;
    buf[ 15 ] = 0x08;
    buf[ 16 ] = 0x08;
    buf[ 17 ] = 0x00;
    buf[ 18 ] = 0x00;
    buf[ 19 ] = 0x00;
*/
    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}

/* read capacity */
readcapacity( id, lun, buf )
int id, lun;
unsigned char *buf;
{
    unsigned char  v[ 10 ], sensbuf[ 256 ];

    v[ 0 ] = C_RDCAP;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;   /*logical block address is 0 */
    v[ 6 ] = v[ 7 ] = 0x00;
    v[ 8 ] = 0x00;              /* PMI bit false */
    v[ 9 ] = 0x00;              /* flag and link false */

    dma = 0;    /* not DMA mode */
    if( hdcmd( id, v, buf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, buf );
        }else{
            printsens( sensbuf );
        }
    }
}

/* Format Unit */
format( id, lun, inlv )
int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];

    v[ 0 ] = C_FMTDRV;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = inlv;
    dma = 0;    /* not DMA mode */
    printf( "\nPhysical format executing. Wait 2 to 5 min.\n" );
    if( hdcmd( id, v, sensbuf ) ){
        if( requestsense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hdcmd( id, v, sensbuf );
        }else{
            printsens( sensbuf );
        }
    }
}

/* Mode select */
modeselectfirst( id, lun )
int id, lun;
{
    unsigned char  v[ 6 ], buf[ 128 ], sensbuf[ 256 ];
    int i, cyl, hd, wp, rwc;

    do{
        printf( "\n\nEnter the cylinder no. of HDD " );
        scanf( "%d", &cyl );
        printf( "\n             Head no. of HDD " );
        scanf( "%d", &hd );
        printf( "\nEnter the 'Write pre compe Cylinder' of\n" );
        printf( "HDD ? ( If it's not necessary, enter the \n" );
        printf( "same No. of Cyl.)                  " );
        scanf( "%d", &wp );
        printf( "\nEnter the 'Reduced write current Cylinder'\n" );
        printf( "of HDD ? ( If it's not necessary, enter \n" );
        printf( "the same No. of Cyl.)              " );
        scanf( "%d", &rwc );
        printf( "\nAre you sure ?  (Y/N)  " );
        scanf( "%s", buf );
    }while( *buf != 'y' && *buf != 'Y' );

    for( i = 0; i < 127; i++ ){
        buf[ i ] = 0;
    }

    v[ 0 ] = C_MODSEL;
    v[ 1 ] = ( lun << 5 ) + 0x10;
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 56;                /* paremeter byte no. */

    buf[  0 ] = 0x00;           /* block headder */
    buf[  1 ] = 0x00;
    buf[  2 ] = 0x00;
    buf[  3 ] = 0x08;

    buf[  4 ] = 0x00;           /* block descriptor */
    buf[  5 ] = 0x00;
    buf[  6 ] = 0x00;
    buf[  7 ] = 0x00;
    buf[  8 ] = 0x00;
    buf[  9 ] = 0x00;
    buf[ 10 ] = ( ( sctsiz == 256 ) ? 0x01: 0x02 ); /* 256 or 512 byte / sector */
    buf[ 11 ] = 0x00;

    buf[ 12 ] = 0x04;           /* page code 4 */
    buf[ 13 ] = 0x0a;           /* parameter byte no. */
    buf[ 14 ] = 0x00;           /* No. of Cyliners */
    buf[ 15 ] = cyl >> 8;
    buf[ 16 ] = cyl & 0x00ff;
    buf[ 17 ] = hd;             /* No. of heads */
    buf[ 18 ] = 0x00;
    buf[ 19 ] = wp >> 8;        /* write precompe start cylinder */
    buf[ 20 ] = wp & 0x00ff;
    buf[ 21 ] = 0x00;           /* reduced write current start cylinder */
    buf[ 22 ] = rwc >> 8;
```

```
	buf[ 23 ] = rwc & 0x00ff;

	buf[ 24 ] = 0x03;		/* page code 3 */
	buf[ 25 ] = 0x16;
	buf[ 26 ] = 0x00;
	buf[ 27 ] = 0x00;
	buf[ 28 ] = 0x00;
	buf[ 29 ] = 0x00;
	buf[ 30 ] = 0x00;
	buf[ 31 ] = 0x00;
	buf[ 32 ] = 0x00;
	buf[ 33 ] = 0x00;
	buf[ 34 ] = 0x00;
	buf[ 35 ] = 0x00;
	buf[ 36 ] = 0x00;
	buf[ 37 ] = 0x00;
	buf[ 38 ] = 0x00;
	buf[ 39 ] = 0x00;
	buf[ 40 ] = 0x00;		/* track skew factor */
	buf[ 41 ] = 0x01;
	buf[ 42 ] = 0x00;
	buf[ 43 ] = 0x00;
	buf[ 44 ] = 0x00;
	buf[ 45 ] = 0x00;
	buf[ 46 ] = 0x00;
	buf[ 47 ] = 0x00;

	buf[ 48 ] = 0x01;	/* page code 1 */
	buf[ 49 ] = 0x06;
	buf[ 50 ] = 0x24;
	buf[ 51 ] = 0x08;
	buf[ 52 ] = 0x00;
	buf[ 53 ] = 0x00;
	buf[ 54 ] = 0x00;
	buf[ 55 ] = 0x00;

	if( hdcmd( id, v, buf ) ){
		if( requestsense( id, lun, sensbuf ) == 6 ){
			/* reset condition */
			hdcmd( id, v, buf );
		}else{
			printsens( sensbuf );
		}
	}
}
/* Format Unit */
formatfirst( id, lun, inlv )
int id, lun, inlv;
{
	unsigned char v[ 6 ], sensbuf[ 256 ];
	int dfno, i;
	struct{
		int cyl;
		int hd;
		int diff;
	}defect[ 100 ];
	char buf[ 1024 ], *ptr;
	printf( "¥nEnter the Defect No. on this HDD ? " );
	scanf( "%d", &dfno );
	do{
		for( i = 0; i < dfno; i++ ){
			printf( "¥nDefect  %3d: Enter the Cylinder ? ", i + 1 );
			scanf( "%d", &( defect[ i ].cyl ) );
			printf( "              Enter the Head No.  ? " );
			scanf( "%d", &( defect[ i ].hd ) );
			printf( "     Enter the position from Index ? " );
			scanf( "%d", &( defect[ i ].diff ) );
		}
		putchar( '¥n' );

		for( i = 0; i < dfno; i++ ){
			printf( "Defect %4d: Cylinder %4d Head%3d %6dBytes¥n",
		i + 1, defect[ i ].cyl, defect[ i ].hd, defect[ i ].diff );
		}
		printf( "¥nAre you sure ? (Y/N)  " );
		scanf( "%s", buf );
	}while( *buf != 'y' && *buf != 'Y' );

	v[ 0 ] = C_FMTDRV;
	v[ 1 ] = ( lun << 5 ) + 0x1e;
	v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
	v[ 4 ] = inlv;

	buf[ 0 ] = 0x00;
	buf[ 1 ] = 0xf1;	/* create p-list */
	buf[ 2 ] = ( dfno * 8 ) >> 8;
	buf[ 3 ] = ( dfno * 8 ) & 0x00ff;

	ptr = &( buf[ 4 ] );
	for( i = 0; i < dfno; i++ ){
		*ptr++ = defect[ i ].cyl >> 8;
		*ptr++ = defect[ i ].cyl & 0x00ff;
		*ptr++ = defect[ i ].hd;
		*ptr++ = defect[ i ].diff >>8;
		*ptr++ = defect[ i ].diff & 0x00ff;
		*ptr++ = 0;
		*ptr++ = 16;
		*ptr++ = 0;
	}
	dma = 0;	/* not DMA mode */
	printf( "¥nPhysical format executing. wait 2 to 5 min. ¥n" );
	if( hdcmd( id, v, buf ) ){
		if( requestsense( id, lun, sensbuf ) == 6 ){
			/* reset condition */
			hdcmd( id, v, buf );
		}else{
			printsens( sensbuf );
		}
	}
}

/* Write 1 Sector */
swrite( id, lun, lsn, buf )
int id, lun;
long lsn;	/* logical sector number */
unsigned char *buf; /* data buffer */
{
	unsigned char v[ 6 ];
	unsigned char sensbuf[ 256 ];

	if( sctsiz == 256 ){
		lsn *= 2L;
	}
	v[ 0 ] = C_WRTSEC;
	v[ 4 ] = ( ( sctsiz == 256 )? 2: 1 );
	v[ 5 ] = 0x00;
	v[ 1 ] = ( lun << 5 ) + ( lsn >> 16 ) & 0x00ff;
	v[ 2 ] = ( lsn >> 8 ) & 0x00ff;
	v[ 3 ] = lsn & 0x00ff;
	dma = 0;	/* write 256 bytes */
	scsimode = 0x0b88;
	if( hdcmd( id, v, buf ) ){
		if( requestsense( id, lun, sensbuf ) == 6 ){
			/* reset condition */
			hdcmd( id, v, buf );
		}else{
			printsens( sensbuf );
		}
	}
}

requestsense( id, lun, buf )
int id, lun;
unsigned char *buf;
{
	/* check error. do request sense. */
	unsigned char v[ 6 ];

	v[ 0 ] = C_REQSEN;
	v[ 1 ] = lun << 5;	/* set Lun */
	v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
	v[ 4 ] = 255;
	dma = 0x00;
	hdcmd( id, v, buf );
	return( buf[ 2 ] );
```

```
}

printsens( buf )
unsigned char *buf;
{
    printf( "Sense key = %02X, Extend sense code = %02X¥n", buf[ 2 ], buf[ 12 ] );
    dump( buf );
    exit( 99 );
}

/* output command to HDD */
hdcmd( id, sc, buf )
int id;
unsigned char *sc, *buf;
{
    int treg;    /* temp. reg. */
    int idb;

    idb = 0x0001 << id;

    /* selection phase */
/*  printf( "Selection.¥n" );*/
    outp( P_DATA, ~(INITBIT+idb) ); /* set target & init. ID */
    outp( P_CNT, ATN+SEL );         /* selection bit true */

    for( treg = 0; treg < 1000; treg ++ ){
     /* wait responding time ( 200us max. ) */
    }

    if( inp( P_SNS ) & BSY ){    /* if BSY is false */
        printf( " couldn't select device.¥n " );
        exit( 99 );
    }
    outp( P_DATA, 0xff );        /* clear data line */
    outp( P_CNT, ATN );      /* selection complete. reset selection bit. */
    /* command phase */
/*  printf( "Command.¥n" );*/
    do{
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait while REQ = false */
        }
        treg &= CD;
    }while( treg ); /* wait until command phase */
    mantfr( sc );   /* output command data */
    /* data transfer phase */
/*  printf( "Data Transfer.¥n" );*/
    while( ( treg = inp( P_SNS ) ) & REQ ){
        /* wait while REQ = false */
    }
    if( ( treg & ( MSG + CD + IO ) ) != 0 ){/* status phase? (MSG=f,CD,IO=t) */
        /* no, data phase. */
        mantfr( buf );
    }
    /* status phase */
/*  printf( "Status.¥n" );*/
    do{
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait while REQ = false */
        }
    }while( ( treg & ( MSG + CD + IO ) ) != 0 ); /* wait until MSG:f CD,IO:t */

    tfrsub( &sttbuf );  /* get Status byte */

    /* message phase */
/*  printf( "Message.¥n" );*/
    do{
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait until REQ = true */
        }
        treg &= ( MSG + CD );
    }while( treg != MSG );
    tfrsub( &msgin );   /* get message byte */
    /* bus free phase */
/*  printf( "Bus free.¥n" );*/
    while( !( inp( P_SNS ) & BSY ) ){
        /* wait until BUSY = false */
    }
    /* disconnect */
    treg = sttbuf;
    if( treg == 0x01 ){
        /* parity error */
        printf( "parity error occured.¥n" );
        exit( 99 );
    }else if( treg == 0x02 ){
        /* check condition */
        return( 1 );
    }else{
        return( treg );
    }
}

/* manual Data transfer routine */
mantfr( buf )
unsigned char *buf;
{
    int treg;   /* temporary register */
    int tphase; /* initial phase storage */

    while( ( treg = inp( P_SNS ) ) & REQ ){
        /* wait until REQ = true */
    }
    tphase = treg & ( MSG + CD + IO );
    do{
        tfrsub( buf++ );
        while( ( treg = inp( P_SNS ) ) & REQ ){
            /* wait until REQ = true */
        }
        treg &= ( MSG + CD + IO );
    }while( treg == tphase );       /* repeat until phase change */

    return( 0 );
}

/* sub routine: 1 byte transfer */
tfrsub( buf )
unsigned char *buf;
{
    int treg;    /* temp. reg. */

    outp( P_DATA,0xff );            /* clear data line */
    while( ( treg = inp( P_SNS ) ) & REQ ){
        /* wait while REQ = false */
    }
    if( ( treg & IO ) == 0 ){
        outp( P_CNT, IOS+DIR+ATN );
        /* I/O IN */
        *buf = ~inp( P_DATA );
        outp( P_CNT, ATN );
    }else{
        /* I/O OUT */
        outp( P_DATA, ~(*buf) );
    }
    outp( P_CNT, ATN + ACK );           /* set ACK bit */
    while( !( inp( P_SNS ) & REQ ) ){
        /* wait until REQ = false */
    }
    outp( P_CNT, ATN );                 /* reset ACK bit */
    outp( P_DATA, 0xff );               /* clear data line */
}

dump( buf )
unsigned char *buf;
{
    int i, j;
    unsigned d;

    printf( "      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF¥n" );

    for( i = 0; i < 256; i += 16 ){
        printf( "%04X: ", i );
        for( j = 0; j < 16; j++ ){
            printf( "%02X ", buf[ i + j ] );
        }
```

〈リスト12〉 SFEHE.C（つづき）

```
        for( j = 0; j < 16; j++ ){
            d = buf[ i + j ];
            if( ( d >= ' ' ) && ( d < 0x7f ) ){
                putchar( d );
            }else{
                putchar( '.' );
                    }
                }
                putchar( '¥n' );
            }
        }
```

〈リスト13〉 EASYHARD.H

```
/* I/O address */

#define P_DATA  0x378
#define P_SNS   0x379
#define P_CNT   0x37a

/* transfer Mode */

#define MSG 0x80
#define REQ 0x40
#define BSY 0x20
#define CD  0x10
#define IO  0x08

#define IOS 0x80
#define DIR 0x20
#define RST 0x08
#define ATN 0x04
#define SEL 0x02
#define ACK 0x01

/* Controler command */

#define C_TSTRDY 0x00
#define C_REZERO 0x01
#define C_REQSEN 0x03
#define C_FMTDRV 0x04
#define C_REDSEC 0x08
#define C_WRTSEC 0x0A
#define C_SEEK   0x0B
#define C_MODSEL 0x15
#define C_MODSEN 0x1a
#define C_RDCAP  0x25

/* SCSI ID */

#define INITID 7
#define INITBIT 0x80
#define TAGTID 0
#define TAGTBIT 0x01

/* Global Variables */

/* data buffer for system */
char sysbuf[ 2048 ];

char sttbuf;    /* buffer for status byte */
char msgin; /* buffer for message byte */
char ints;  /* interrupt status */
int dma;
int dmamode;
int scsimode;
```

SPC(MB89352)の解説

宇野沢成夫

富士通のSCSIプロトコル・コントローラにはいくつかのバリエーションがありますがMB89352は同期転送はサポートしていないもののドライバ/レシーバを内蔵しており，非常に簡単な回路でパソコンなどのSCSIインターフェースを作成することができます．

MB89352は8バイトFIFOバッファおよびそれをコントロールする転送バイト・カウンタを内蔵しており，簡単な手順で高速なデータ転送が可能になっています．

パッケージは48ピンDIPと48ピンQFPの2種類があります(図A)．

MB89352のデータ・シートにも記載されていますが，ここではSCSIプロトコル・コントローラを略してSPCと呼びます．SPCの場合，イニシエータとターゲット両方に使用できるのですがここではイニシエータとして使う場合に限って説明します．

● MB89352のレジスタ

MB89352には14バイトのアクセス可能なレジスタがあります(表A)．

(1) BDIDレジスタ

BDIDレジスタは，SCSI IDを書き込みます．読み込み時は書き込まれたIDに対応する1ビットがセットされています．

(2) SCTLレジスタ

SCTLレジスタはSPCの動作を設定するレジスタで，通常BDIDレジスタとともに初期化のときに設定します．

ビット7のReset & Disableを“1”にセットすることでSCSIバスと論理的に切り離された状態になり，SPC内部のリセットを行います．また，CPU側よりハードウェア・リセットを行われた際“1”にセットされるのでSPCを使うときにBDIDレジスタへSCSI IDをセットした後，“0”にセットします．

ビット6のControl ResetはSPCのデータ転送制御回路に対するリセットで，“1”の間リセットが実行されますが，SCSIバスと結合中でも結合関係は保持されます．

ビット5を“1”にした場合SPCは診断モードと呼ばれるモードになりSCSIとは論理的に切り離されSDGCレジスタにより擬似的にSCSI上の動作を実行することができます．

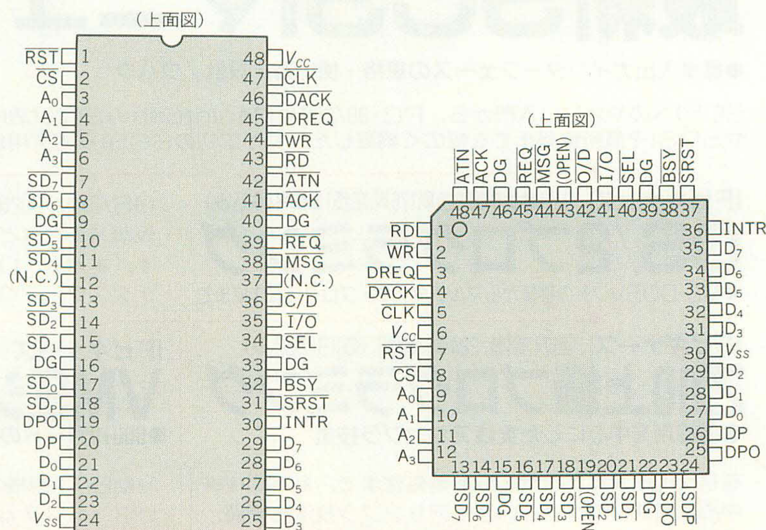
ビット4はアービトレーション，ビット3はパリティ・チェックを実行するかどうか指定するビットです．

ビット2はセレクションに応答するかどうかを指定するビットなのですが，イニシエータのみの場合このビットを“0”にセットします．

ビット1はリセクションに応答するかどうかを指定するビットで，ディスコネクトをドライバがサポートする場合にはこのビットを“1”にします．

ビット0はSPCからインタラプトを許可するかど

〈図A〉
MB89352のピン配置



(a) DIP端子配列図

(b) フラット・パッケージ端子配列図

うかを指定するビットで，“1”にした場合はインタラプトを許可します．

(3) SCMDレジスタ

SPCの場合，SCSIバスへなにかの動作をするためにはコマンドをこのSCMDレジスタへ発行するという形態をとっています．ビット7，6，5へはコマンドを書き込みます．コマンドは**表B**のような種類があり，このレジスタへ書き込まれた時点で実行されます．

ビット4はSCTLレジスタが“0”のときにこのビットを“1”にすることでSCSIバスのRST信号をアサートすることができます．

ビット3を“1”にしてSet ACK/REQとReset ACK/REQコマンドを用いてマニュアル転送を行う場合はSPC内の8バイトFIFOレジスタの内容は保

〈表B〉 SCMDレジスタ

ビット7	ビット6	ビット5	コマンド
0	0	0	Bus Release
0	0	1	Select
0	1	0	Reset ATN
0	1	1	Set ATN
1	0	0	Transfer
1	0	1	Transfer Pause
1	1	0	Reset ACK/REQ
1	1	1	Set ACK/REQ

〈表A〉 内部レジスタのアドレスおよびビット割り付け

アドレス	名称 (略称)	R/W	レジスタ・ビット 7	6	5	4	3	2	1	0	パリティ
0	Bus Device ID (BDID)	R	#7	#6	#5	#4	#3	#2	#1	#0	0
		W	—					ID4	ID2	ID1	—
1	Spc Control (SCTL)	R/W	Reset & Disable	Control Reset	Diag Mode	ARBIT Enable	Parity Enable	Select Enable	ReSelect Enable	INT Enable	P
2	Comand (SCMD)	R/W	Command Code			RST Out	Intercept Xfer	Transfer Modifier PRG Xfer	0	Term Mode	P
4	Interrupt Sense (INTS)	R	Selected	ReSelected	Disconnected	Command Complete	Service Required	Time Out	SPC Hard Error	Reset Condition	P
		W	Reset Interrupt								—
5	Phase Sense (PSNS)	R	REQ	ACK	ATN	SEL	BSY	MSG	C/D	I/O	P
	SPC Diag. Control (SDGC)	W	Diag REQ	Diag ACK	Xfer Enable	—	Diag BSY	Diag MSG	Diag C/D	Diag I/O	—
6	SPC Status (SSTS)	R	Connected INIT	TARG	SPC BSY	Xfer in Progress	SCSI RST	TC=0	DREG Status FULL	EMPTY	P
7	SPC Error Status (SERR)	R/W	Data Error SCSI	SPC	Xfer Out	0	TC P-Error	0	Short Period	0	P
8	Phase Control (PCTL)	R/W	Bus Free Interrupt Enale	0				Transfer Phase MSG Out	C/D Out	I/O Out	P
9	Modified Byte Counter (MBC)	R	0				MBC 3	2	1	0	P
A	Data Register (DREG)	R/W	Internal Data Register (8バイトFIFO) 7	6	5	4	3	2	1	0	P
B	Temporary Register (TEMP)	R	Temporary Data (Input : From SCSI) 7	6	5	4	3	2	1	0	P
		W	Temporary Data (Output : To SCSI) 7	6	5	4	3	2	1	0	P
C	Transfer Counter High (TCH)	R/W	Transfer Counter High (MSB) 23	22	21	20	19	18	17	16	P
D	Transfer Counter Mid (TCM)	R/W	Transfer Counter Middle (2ndバイト) 15	14	13	12	11	10	9	8	P
E	Transfer Counter Low (TCL)	R/W	Transfer Counter Low (LSB) 7	6	5	4	3	2	1	0	P

(注) 1．書き込み動作において—が記入されているビットは，1，0のどちらを書き込んでもよいことを示します．
2．読み出し専用レジスタへの書き込み動作は無視されます．
3．読み出し時において0と示されているビットは必ず0が読み出されます．

存されます．これは，データ転送フェーズの最中にメッセージ・フェーズへのフェーズの変更によるService Require割り込みの処理などに用いられます．

ビット2はTransferコマンドを実行するときにDMACを使用するためのDREQを出すかどうかを指定するビットで“0”のときにDREQを出します．

ビット0はTransferコマンド実行中に転送バイト・カウンタがゼロになった場合の処理を指定するビットで，“0”を指定した場合はバイト・カウンタがゼロになった時点でTransferコマンドを終了します．

“1”を指定した場合，アウトプット動作中であれば00Hのデータをフェーズの変更があるまで転送しつづけます．また，インプット動作中であれば受領データをフェーズ変更があるまで無視します．これはPadding転送と呼び，SPC内部で実行されるのでDMA転送時のDREQやプログラム転送時のDREGレジスタ・アクセス要求のINTR信号は出されません．

(4) INTSレジスタ

割り込み原因の表示およびリセットを行うためのレジスタです．SPCからの割り込みはビット0のReset Conditionを除いてSCTLレジスタのビット0によりマスク可能です．

割り込みのリセットはこのレジスタに表示される割り込み原因のビット位置に“1”を書き込むことにより行います．ビット7はSCSIバス上の他のイニシエータよりセレクション・フェーズによりセレクトされたことを示します．この場合SPCはBus Releaseコマンドを発行するかリセット・コンディションを検出するまでターゲット・モードになります．

ビット6はSCSIバス上のほかのターゲットよりセレクション・フェーズによりセレクトされたことを示します．

ビット5はPCTLレジスタのビット7が“1”のときSCSIバスが，バス・フリーへ移行したことを示します．

ビット4はSelectまたはTransferコマンドの動作が終了したことを示します．

ビット3はTransferコマンド動作中に転送フェーズがPCTLレジスタのビット2，1，0に指定されているものと違うフェーズの場合を示します．

ビット2はSelectコマンドによりセレクション・フェーズを実行したが指定された監視時間を経過してもターゲットから応答がなかったことを示します．

ビット1はSPCがSERRレジスタのビット3，1に表示されるエラーがあることを示します．

ビット0はSCSIバス上でリセット・コンディションを検出したことを示します．この割り込みを検出した場合SSTレジスタのビット3が“0”であることを確認した上で，この割り込みをリセットします．

(5) PSNSレジスタ

SCTLレジスタのビット5が“0”のときはSCSIバス上の信号線の状態を示し，“1”のときはSPCから送出する信号の状態を示します．

(6) SDGCレジスタ

SCTLレジスタのビット5が“1”のとき(診断モード)SCSIバス上の信号線を擬似的に動作させるために使用します．

(7) SSTSレジスタ

SPCの内部のステータスを表示するレジスタです．ビット7，6はSPCとSCSIの結合状態を示します．

ビット7	ビット6	状　態
0	0	非結合中
1	0	イニシエータとして結合中
0	1	ターゲットとしと結合中
1	1	未定義

ビット5はコマンドの実行中または実行待ちの状態であることを示します．

ビット4はハード転送中またはSCSIバス上で転送フェーズが要求されていることを示します．

ビット3はSCSIバス上のRST信号の状態を示します．

ビット2は転送バイト・カウンタがゼロであることを示します．

ビット1，0はFIFOバッファの状態を示します．

プログラム転送時はこのビットの値を見ることにより，データ・バッファへのアクセス・タイミングを知ることができます．

ビット1	ビット0	状　態
0	0	1～7バイトのデータを保持している
0	1	空の状態
1	0	8バイトのデータを保持していて空なし
0	1	未定義

(8) SERRレジスタ

SPC内部で検出されたエラーの詳細を表示するレジスタです．

ビット7，6はSCSIの転送フェーズを実行中にパリティ・エラーが検出されたことを示します．

ビット3は転送バイト・カウンタの減算動作でパリティ・エラーが検出されたことを示します．

ビット1はSCSIバス上のREQ/ACK信号がSPCの追従範囲を超えた(速すぎる)周期で入力されたことを示します．

(9) PCTLレジスタ

ビット7を“1”に設定したとき，SCSIバス上でバス・フリー・フェーズを検出すると，Disconnect割り込みが発生します．Selectコマンド発行時およびDis

connect 割り込みをリセットするときは必ずこのビットを“0”にセットします．

なお，Select コマンドを実行する場合はコマンドの発行に先だって，00H を設定するとセレクション・フェーズ，01H を設定するとリセクション・フェーズを実行します．また，SCSI バスと結合中の場合は実行予定のフェーズをビット 2，1，0 へ指定します．

⑽ MBC レジスタ

SPC の FIFO バッファと MPU またはメモリとのデータ転送数を制御するカウンタです．

⑾ DREG レジスタ

SPC の FIFO データ・バッファで Transfer コマンド実行時このレジスタを介してデータを転送します．

⑿ TEMP レジスタ

ハード転送以外の場合に SCSI データ・バスを制御するための送信/受信の独立したレジスタです．

⒀ 転送バイト・カウント・レジスタ(TCH，TCM，TCL)

3 バイト構成の減算カウンタでハード転送動作実行時に 1 バイトのデータを転送するごとに 1 バイト減らされ，転送すべき残りバイト数を表します．

また，Select コマンドの実行時には応答待ち時間監視用のタイマとして動作します．

● MB89352 のコマンド

MB89352 の SCSI バスへの動作はコマンドの実行(SCMD レジスタのへの書き込み)により行われます．

(1) Bus Release コマンド

ターゲットとして動作中のときはバス・フリー・フェーズへ移行しますが，イニシエータではバス・フリー待ちの Select コマンドを解除する場合に用います．

(2) Select コマンド

このコマンドを受け取るとバス・フリー・フェーズを検出後 SCTL レジスタのビット 4 が“1”であればアービトレーションを行いバス使用権を獲得した場合 PCTL レジスタのビット 0 が“0”の場合セレクション，“1”の場合リセレクション・フェーズを実行します．SCTL レジスタのビット 4 が“0”であればバス・フリー・フェーズを検出後にセレクション・フェーズを実行します．

このコマンドを発行する前に TCH，TCM レジスタへセレクション・フェーズにおける応答時間(T_{sl})を TCL レジスタへ，バス・フリー検出後アービトレーションおよびセレクション・フェーズへ移行するまでの時間(T_{wait})，TEMP レジスタへターゲットおよびイニシエータのデバイス ID を設定します．

また，セレクション・フェーズを実行する際に SCSI バスへ ATN 信号を送出する必要がある場合は，前もって Set ATN コマンドを実行しておきます．

▶ TCH，TCM へ書き込む値 N を算出する式

$N \neq 0$ のとき $T_{sl}=(N\times256+15)\times T_{clf}\times2$

$N=0$ のとき $T_{sl}=\infty$

(T_{clf} は SPC の CLK 端子に供給するクロックの周期)

▶ TCL へ書き込む値の推奨値

T_{clf}(ns)	TCL	T_{wait}[平均(ns)]
125～180	04H	1250～1980
140～200	03H	1260～2000

(3) Set ATN および Reset ATN

SCSI バス上の ATN 信号をセット/リセットするコマンドです．

(4) Transfer コマンド

SCSI の情報転送フェーズを実行するコマンドで，これによって行われる転送をハード転送と呼びます．ハード転送はさらに SCMD レジスタのビット 2 によりプログラム転送モードと DMA 転送モードに分かれます．このコマンドでは SCSI バス上の REQ/ACK のハンドシェイクなどは SPC が制御するので，コマンド発行前に転送バイト・カウンタ(TCH，TCM，TCL)，PCTL レジスタの実行フェーズを示すビット・パターンの初期設定が必要になります．イニシエータとして動作しているときは，REQ 信号を受領したときに設定したフェーズが一致した場合に転送を開始します．転送終了は Disconnect 割り込みの発生時，Padding 転送モードでない場合は転送バイト・カウンタがゼロになった時点，Padding 転送時はフェーズが不一致した時点で終了します．

プログラム転送モードでは DREG レジスタを介してデータ転送を行いますが，DREG レジスタの状態を知る方法としては 2 種類あります．ひとつは SSTS レジスタを読み出すことによりポーリングを行う方法，もうひとつは SDGC レジスタのビット 5 を“1”にすることによって DREG レジスタが FULL か EMPTY かを INTR による割り込みによって知る方法です．

DMA 転送の場合は DREQ 信号とその応答による DACK によって行いますが，DREQ 信号は転送するデータが DREG レジスタ内に存在する間は常にアサートされた状態になるので，エッジ・トリガを必要とする DMAC を使用する場合は若干の回路修正が必要になります．

(5) Set ACK/REQ および Reset ACK/REQ コマンド

SCSI バス上の REQ/ACK 信号線のハンド・シェイクをマニュアルで転送する場合のコマンドです．

この場合もあらかじめ PCTL レジスタへ実行フェーズを示すビット・パターンをあらかじめ設定しておかなければなりません．データは TEMP レジスタを介して行われます．

第7章

SCSI2の概要—拡張と変更を中心に

もう一部ではSCSI3の話も．乗り遅れるな！

菅谷誠一

1986年にANSIでSCSI（Small Computer System Interface, ANSI規格番号：X3.131-1986）の規格化が完了して以来，中小型システム分野を中心にSCSIの適用が急速に進展し，これらに接続される小型ハードディスク装置，カートリッジ磁気テープ装置，光ディスク装置などの入出力装置の標準I/Oインターフェースとして SCSIは完全に定着しています．

ANSIではSCSIの規格がまとまった直後から，さらに高機能化および高性能化を図るための検討を行ってきました．この拡張仕様はSCSI2と呼ばれ，各種システムからの高性能化の要求に応えるとともに，適用可能分野の拡大をねらったものです．図1にSCSIの標準化の推移を示します．

SCSI2は，まだ正式にANSI規格になったものではありませんが，1990年3月にANSIの作業部会（X3T9.2）から最終規格案（文書番号：X3T9.2/86-109 Revision 10c）が発行され，仕様のほとんどが明確になっています．また，すでにこの仕様を適用したハードディスク装置などが登場してきています．今後，正式に規格化が完了するまでの間に，仕様の追加などが行われる可能性はありますが，ここでは現在明らかになっているSCSI2の仕様に基づいて従来のSCSI仕様との相違および拡張された機能について，ハードディスク装置を中心に解説します．

なお，区別を明確にするために，従来のSCSI仕様をSCSI1と呼ぶことにします．

SCSI2の概要

SCSI2の開発のねらいはSCSI1との継続性（下方互換性）を維持しながらの機能拡張および高性能化であるといえます．それらの主要項目の概要はつぎのとおりです．

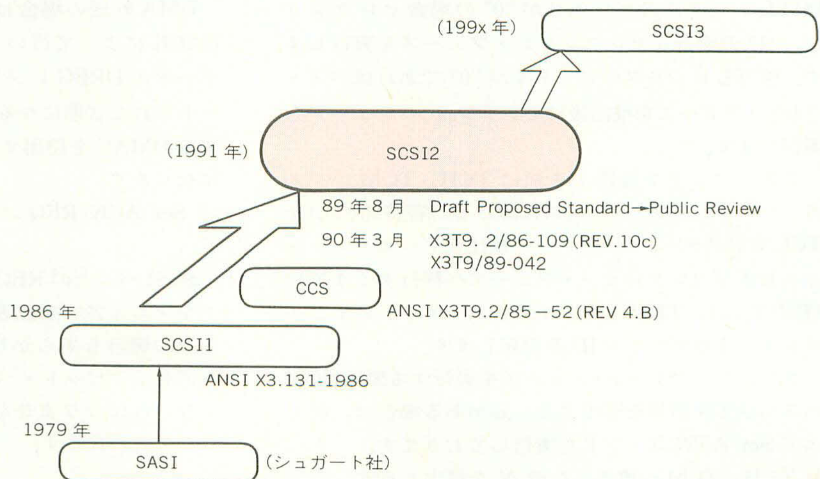
● SCSI1との互換性

SCSI1では表1に示したように，オプションまたは拡張機構として規定されている機能の組み込みの度合いによってレベル0〜2の機能レベルが定義されていました．また，SCSI1のベースとなったSASIとの互換性が保たれていました．

SCSI2ではSCSI1でオプションとなっていた，データ・バス・パリティ，アービトレーション・フェーズ，アテンション・コンディションが必須の機能になったため，データ・バス・パリティを使用しているレベル2のSCSI1適用環境との接続互換性のみが確保されています．

したがってSCSI2適用の装置は，この条件を満た

〈図1〉SCSI 標準化の推移



している SCSI1 の装置と同一のバス上に共存させることができます．SASI との整合性は発展的に放棄されていますが，SCSI1 の標準化完了後に製品化されたほとんどの SCSI 装置はレベル 2 の機能を満たしているので，実質的には SCSI1 との互換性が確保されているといえます．

● サポート・デバイスの拡充

表 2 に示すように，あらたに 4 種のデバイス・タイプが追加され，各々に標準のコマンド・セットが定義されています．

また，SCSI1 でリード・オンリ・デバイスと呼ばれていたデバイス・タイプは CD-ROM デバイスと呼称が変わり，装置の性格付けを明確にしました．

● 高性能化

データ転送の高速化を図るためにふたつの手法が導入されました．

ひとつはインターフェース・ケーブルを追加することでデータ・バスを 2 バイト幅または 4 バイト幅に拡張して高速データ転送を可能にするもので，WIDE SCSI と呼ばれます．

もうひとつの手段は同期モードのデータ転送フェーズのタイミング規約を高速化し，1 バイト幅のデータ・バスで最大 10M バイト/s までのデータ転送を可能とするもので，FAST SCSI と呼ばれます．

これにより，SCSI1 では 4M～5M バイト/s(同期モード)が上限であったデータ転送性能が 1 バイト・バスで最大 10M バイト/s まで，4 バイト・バスで最大 40M バイト/s まで仕様上可能になりました．

● 高機能化

まず最初にコマンド・キューイングの機能が追加されたことがあげられます．この機能を使用することでひとつのコマンドの実行が終了するのを待たずに同一のイニシエータから同一のロジカル・ユニットに対して複数個(最大 256 個)のコマンドを発行することができます．

ターゲットは受け取った複数のコマンドを待ち行列(キュー)に入れ，ロジカル・ユニット上でそれらのコマンド列をもっとも効率よく高速に実行するようにコマンドの実行順序を変更することもできます．

また，キャッシュ・メモリを備えて実効入出力性能の向上を図る装置に対してイニシエータからそのキャッシュ機構の動作を制御するためのコマンド・セットやパラメータなどの定義が追加になっています．

これらのふたつの追加機能はシステム内での入出力動作の性能向上を図ることをねらっています．

その他の主要な追加機能としてはロジカル・ユニット上で非同期に発生した事象(たとえば周辺装置の電源が単独で投入され装置がレディ状態になったときなど)をターゲットからイニシエータに通知するための AEN(Asynchronous Event Notification)や入出力

〈表 1〉 SCSI1 の機能レベル

機能項目 ＼ 機能レベル	イニシエータとしての必要条件			ターゲットとしての必要条件		
	レベル 0	レベル 1	レベル 2	レベル 0	レベル 1	レベル 2
アービトレーション・フェーズ		○	○			
COMMAND COMPLETE メッセージ	○	○	○	○	○	○
MESSAGE REJECT メッセージ			○	△	△	○
IDENTIFY メッセージ			○			○
DISCONNECT メッセージ			○			
SAVE DATA POINTER メッセージ			○			
すべての M(サポート必須)レベルのコマンド				○	○	○
すべての E(拡張機能)レベルのコマンド						○
GOOD ステータス	○	○	○	○	○	○
CHECK CONDITION ステータス	○	○	○	○	○	○

○：サポートする必要がある
△：COMMAND COMPLETE 以外のメッセージがあるときにはサポートする必要がある

〈表 2〉 デバイス・タイプ

範囲	デバイス・タイプ・コード(16進)	デバイス・タイプ
SCSI1の範囲 / SCSI2の範囲	00h	ダイレクト・アクセス・デバイス
	01h	シーケンシャル・アクセス・デバイス
	02h	プリンタ・デバイス
	03h	プロセッサ・デバイス
	04h	単一書き込み(Write-once)デバイス
	05h	CD-ROM デバイス(SCSI1：リード・オンリ・デバイス)
	1Fh	未定義のデバイス・タイプ
SCSI2の範囲	06h	スキャナ・デバイス
	07h	光メモリ(Optical Memory)・デバイス
	08h	メディア・チェンジャ(Media Changer)・デバイス
	09h	通信(Communications)デバイス

デバイス・タイプ・コードはスタンダード INQUIRY データ(図 21)に表示される値

動作でエラーが発生したときなどにそのコマンドを発行したイニシエータがエラー情報の収集やエラーの回復処理を実行する間，他のイニシエータからのそのロジカル・ユニットへのアクセスを一時的に排除する(排他制御)手順であるECA(Extended Contingent Allegiance)があります．

● 互換性確保の強化

SCSI1のときには多くの機能がオプションあるいは拡張仕様として規定され装置への組み込みが義務づけられていなかったり，仕様の記述に曖昧な点がありました．このため，メーカ間で実装置への組み込み仕様に差異が生じ，互換性に問題を残しました．

そこでANSIの作業部会ではもっとも需要の大きいダイレクト・アクセス・デバイスについてコマンド体系などの論理仕様の標準形を定めたCCS(Common Command Set)仕様を1986年に発表し，装置間の互換性の確保を図りました．CCS仕様はANSIの規格にはなっていませんが，その後のハードディスク装置のほとんどはこの仕様に基づいたものとなっています．また，その他のデバイス・タイプについてもCCSの考え方が組み込み仕様のモデルとなってきました．

SCSI2ではこのCCSの思想をすべてのデバイス・タイプについて適用し，組み込み仕様の必要条件を明確にしています．また，バス・フェーズの動作規定，コマンド仕様，例外条件の処理などの記述を明確にするとともにエラー・リカバリ情報の標準化を図るためにセンス・データ上に表示されるエラー原因を表すコード(ASC：Additional Sense Code, ASCQ：Additional Sense Code Qualifier)をすべてのデバイス・タイプについて体系化しています．

*　　　*

以下ではSCSI2で変更あるいは追加になった機能を中心に詳細を解説しますが，基本的にはSCSI1の発展形なのでSCSI1の知識・経験を生かせば，容易にSCSI2への移行ができると思われます．

物理仕様

信号伝送系にはSCSI1と同様にシングル・エンド型バスとディファレンシャル型バスが規定されています．電気的条件やタイミング条件，最大ケーブル長も基本的に同一です．

● Bケーブル

インターフェース・ケーブルは50信号線のAケーブルと68信号線のBケーブルとから構成されます．基本のAケーブルはSCSI1と同一の構成と機能を持っています．Bケーブルはオプションであり，WIDE SCSIでデータ・バス幅を2バイトまたは4バイトに拡張するときだけに必要となるもので，最大3バイトまでの拡張データ・バスとBケーブル専用のデータ転送制御信号であるREQB，ACKB，およびターミネータ用電源などが定義されています．

現在の仕様ではBケーブルはWIDE転送専用データ・バスとしての機能しかなく，しかもWIDE SCSIを組み込むためにはふたつのインターフェース・コネクタが必要になるので小型の機器に適した仕様とはいえません．

ANSIでは，1本のケーブル(68ピン：2バイト転送，110ピン：4バイト転送)でデータ・バスを拡張し，さらに追加されたデータ・バス・ビットをSCSI IDの拡張に利用してバス上に接続できるデバイスの数を増加させる改善案を検討していますが，SCSI2の最終規格に盛り込まれるかどうかは不明です．

● ケーブル特性

信号伝送系の構成がSCSI1と同一であり，ケーブルに要求される特性条件も基本的にSCSI1と変わりはありません．

しかし，FAST SCSIを用いる場合には厳しいタイミング条件が要求されるため，以下の電気的条件を満たすシールド型ケーブルが必要であるとされています．

・特性インピーダンス：90～132Ω
・信号減衰特性：5MHzで0.095dB/m以下
・信号線間の伝搬遅延時間差：0.20ns/m以下
・直流抵抗：20°Cで0.230Ω/m以下

● コネクタ

インターフェース・コネクタはつぎのように体系化されます．1.27mmピッチが追加され，装置の小型化への対応が図られています．

▶ Aケーブル：50コンタクト
・ノン・シールド型：(1) 1.27mmピッチ高密度型
(2) 2.54mmピッチ低密度型
・シールド型：(3) 1.27mmピッチ高密度型
(4) 2.16mmピッチ・リボン型

▶ Bケーブル：68コンタクト
・ノン・シールド型：(5) 1.27mmピッチ高密度型
・シールド型：(6) 1.27mmピッチ高密度型

図2～図5に各コネクタの外観を示します．

このうちAケーブル用の低密度型ノン・シールド・コネクタ(図3)とリボン型シールド・コネクタ(図5)はSCSI1と物理的互換性を持っています．なお，SCSI2で新規に規定された高密度型コネクタは装置側がソケット(レセプタクル)で，ケーブル側がプラグ(ピン・コンタクト)になっているので注意してください．

各コネクタ上でのシングル・エンド型バスとディファレンシャル型バスの信号割り付けを表3～表6に示します．

Aケーブルは SCSI1互換の信号割り付けになって

いますが，TERMPWR 信号(ターミネータ用電源)の両サイドのピンは“リザーブ”(従来はグラウンド)になりました．これらのピン(将来の規格で使用される可能性がある)はターミネータ・モジュールあるいはバス端に接続される装置内でグラウンドに接続します．それ以外の装置ではオープンにすることが原則ですが，グラウンドに接続することも許容されています．

● 電気的条件

ドライバ/レシーバおよびターミネータの電気的条件に関する規定は以下に述べる事項をのぞき，原則的に SCSI1 と同一です．表 7 に電気的条件の一覧と規定の相違を示します．

▶ シングル・エンド型バスのターミネータ

シングル・エンド型バスは元々ノイズ・マージンが厳しく，システムを構成するときのネックになっていました．信号伝送特性を改善するために図 6 に示すターミネータ回路が新たに追加されました．SCSI1 と同一の抵抗分圧型（220-330Ω）ターミネータも併記されており，必要に応じてどちらかを選択して使用することができます．

▶ ターミネータ用電源の供給

SCSI1 では TERMPWR の使用はオプションとなっていましたが，SCSI2 ではターミネータには必ず TERMPWR(B ケーブルは TERMPWRB)から電源を供給することとされています．そして通常イニシエータとして動作する装置がその電源を供給する必要があります．ターゲットとしてだけ動作する装置または COPY コマンドや AEN のときにだけ一時的にイニシエータとして動作する装置は，電源を供給できるようにしても構いたせんが，その義務はありません．

● SCSI バス信号

SCSI1 との相違はつぎのとおりです．

▶ データ・バス・パリティ

パリティ・ビット(奇数パリティ)を使用することが必須条件になりました．

▶ SEL 信号

SEL 信号の駆動方法は OR-tied 型とすることになりました．しかし，この規定は将来の規格拡張に備えたもので Non-OR-tied で駆動している従来の装置と混在接続しても問題はありません．

▶ B ケーブル信号

WIDE SCSI 用の 3 バイトの拡張データ・バスと転送制御用の REQB，ACKB 信号が追加されました．B ケーブル上のデータ転送のタイミング(ハンドシェイク)は専用の REQB/ACKB によって制御されます．データ・バス・ビットと各バイトごとのパリティ・ビットとの対応はつぎのとおりです．

- DB_{15}～DB_{8} ：DB_{p1}
- DB_{23}～DB_{16}：DB_{p2}
- DB_{31}～DB_{24}：DB_{p3}

● SCSI バス・タイミング

タイミング規約の一覧を表 8 に示します．FAST SCSI のタイミングが規定された他，今までは各装置の組み込み仕様に依存していた電源投入後またはリセット・コンディション解除後に特定のコマンドに応答できるようになるまでの時間規定(Power-On to Selection Time, Reset to Selection Time：ただし，推奨値として)などが追加されました．

また，Arbitration Delay の規定値が変更になっていますが，これは従来の規定値(2.2μs)が最大ケーブル長(25m)で両端の装置がアービトレーション・フェーズに競合したときに矛盾(実動作ではほとんど問題とならない)していたために修正したものです．

バス・プロトコル

バス・フェーズやバス・コンディションの動作規定については，アービトレーション・フェーズ，リセレクション・フェーズおよびアテンション・コンディションがオプションではなくなりました．

また，以下に述べる変更や追加がありますが，プロトコル規定とタイミング条件は SCSI1 と同等で，同一バスへの混在接続を可能としています．

● セレクション・フェーズ

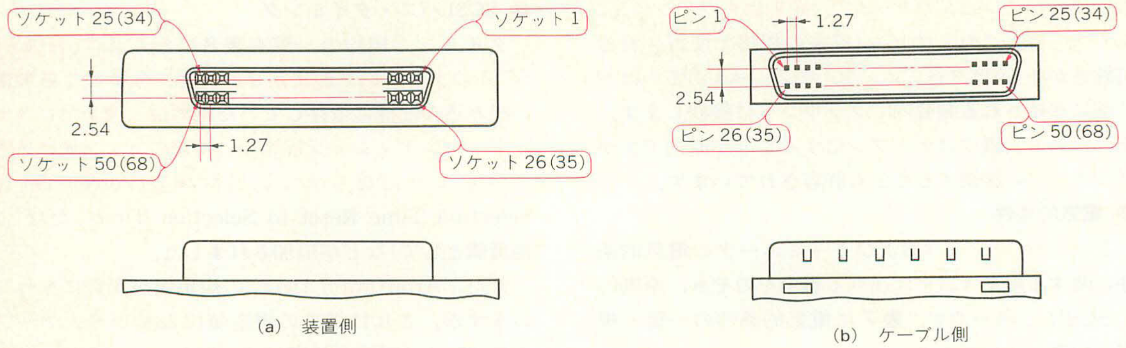
シングル・イニシエータ・オプションの規定は削除されました．したがってイニシエータはセレクション・フェーズ実行時に自分自身とターゲットの SCSI ID をデータ・バス上に送出する必要があります．

また，イニシエータはセレクション・フェーズでは必ずアテンション・コンディションを生成する必要があります．つまり通常のシーケンスでは IDENTIFY メッセージを送信してロジカル・ユニットの番号(LUN)などを通知します．これらは SCSI1 ではオプションとして規定されていたので，これらの機能がサポートされていない SCSI デバイスとの接続を可能にするには，ターゲットとして動作する装置では SCSI1 の規定(イニシエータ ID の通知またはアテンション・コンディションがない場合)でも動作できるように考慮しておく必要があります．

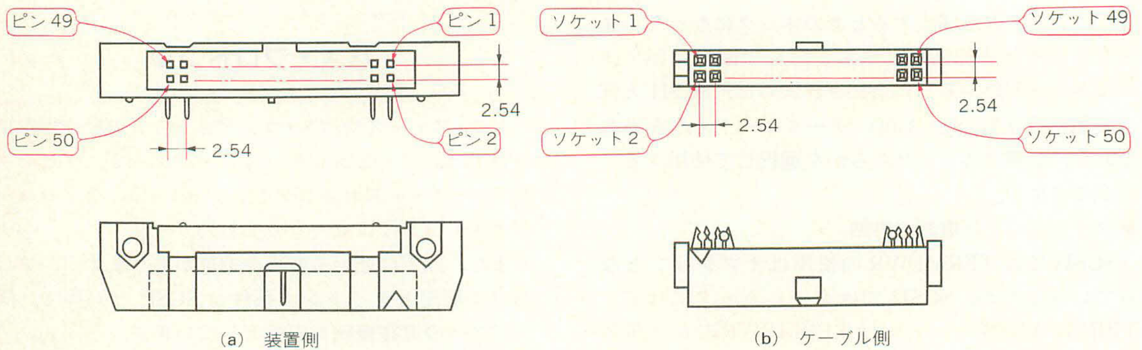
● アテンション・コンディション

SCSI1 ではターゲットがアテンション・コンディションに対して応答すべきタイミングが非常に曖昧に記述されていましたが，SCSI2 では各フェーズごとにアテンション・コンディションに対する応答規定が明記されました．ターゲットは少なくともアテンション・コンディションが生成されたバス・フェーズの直後にはメッセージ・アウト・フェーズに移行してイニシエータからのメッセージを受け取らなければなりません．

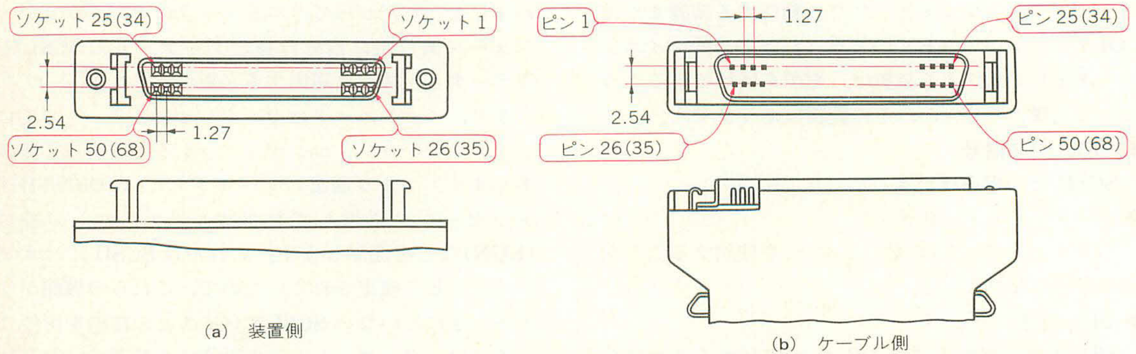
〈図 2〉 高密度型ノンシールド・コネクタ(A/B ケーブル)



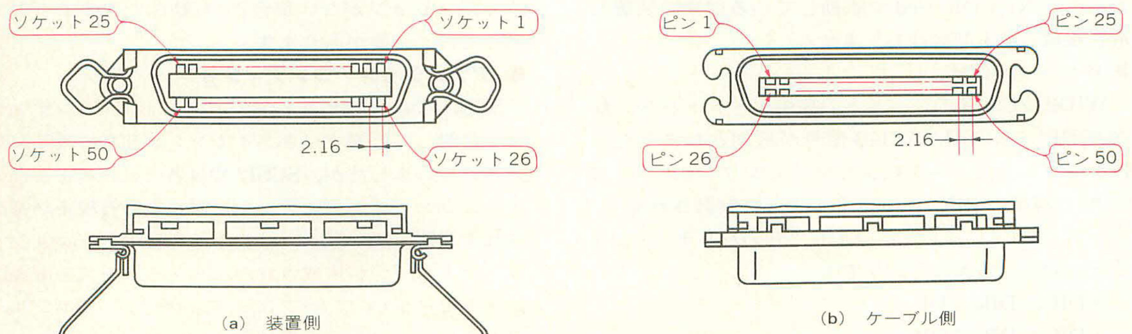
〈図 3〉 低密度型ノンシールド・コネクタ(A ケーブル)



〈図 4〉 高密度型シールド・コネクタ(A/B ケーブル)



〈図 5〉 リボン型シールド・コネクタ(A ケーブル)



〈表 3〉シングル・エンド型バス-A ケーブル信号割り付け

コンタクト番号		信号	信号	コンタクト番号	
01	01	GND	$-DB_0$	02	26
02	03	GND	$-DB_1$	04	27
03	05	GND	$-DB_2$	06	28
04	07	GND	$-DB_3$	08	29
05	09	GND	$-DB_4$	10	30
06	11	GND	$-DB_5$	12	31
07	13	GND	$-DB_6$	14	32
08	15	GND	$-DB_7$	16	33
09	17	GND	$-DB_P$	18	34
10	19	GND	GND	20	35
11	21	GND	GND	22	36
12	23	(リザーブ)	(リザーブ)	24	37
13	25	(オープン)	TERMPWR	26	38
14	27	(リザーブ)	(リザーブ)	28	39
15	29	GND	GND	30	40
16	31	GND	− ATN	32	41
17	33	GND	GND	34	42
18	35	GND	− BSY	36	43
19	37	GND	− ACK	38	44
20	39	GND	− RST	40	45
21	41	GND	− MSG	42	46
22	43	GND	− SEL	44	47
23	45	GND	− C/D	46	48
24	47	GND	− REQ	48	49
25	49	GND	− I/O	50	50

2.54mmピッチ低密度型コネクタ（図3）のコンタクト番号

1.27mmピッチ高密度型（図2,4），リボン型シールド・コネクタ（図5）のコンタクト番号

〈表 5〉ディファレンシャル型バス-A ケーブル信号割り付け

コンタクト番号		信号	信号	コンタクト番号	
01	01	GND	GND	02	26
02	03	$+DB_0$	$-DB_0$	04	27
03	05	$+DB_1$	$-DB_1$	06	28
04	07	$+DB_2$	$-DB_2$	08	29
05	09	$+DB_3$	$-DB_3$	10	30
06	11	$+DB_4$	$-DB_4$	12	31
07	13	$+DB_5$	$-DB_5$	14	32
08	15	$+DB_6$	$-DB_6$	16	33
09	17	$+DB_7$	$-DB_7$	18	34
10	19	$+DB_P$	$-DB_P$	20	35
11	21	DIFFSENS	GND	22	36
12	23	(リザーブ)	(リザーブ)	24	37
13	25	TERMPWR	TERMPWR	26	38
14	27	(リザーブ)	(リザーブ)	28	39
15	29	+ ATN	− ATN	30	40
16	31	GND	GND	32	41
17	33	+ BSY	− BSY	34	42
18	35	+ ACK	− ACK	36	43
19	37	+ RST	− RST	38	44
20	39	+ MSG	− MSG	40	45
21	41	+ SEL	− SEL	42	46
22	43	+ C/D	− C/D	44	47
23	45	+ REQ	− REQ	46	48
24	47	+ I/O	− I/O	48	49
25	49	GND	GND	50	50

2.54mmピッチ低密度型コネクタ（図3）のコンタクト番号

1.27mmピッチ高密度型（図2,4），リボン型シールド・コネクタ（図5）のコンタクト番号

〈表 4〉シングル・エンド型バス-B ケーブル信号割り付け

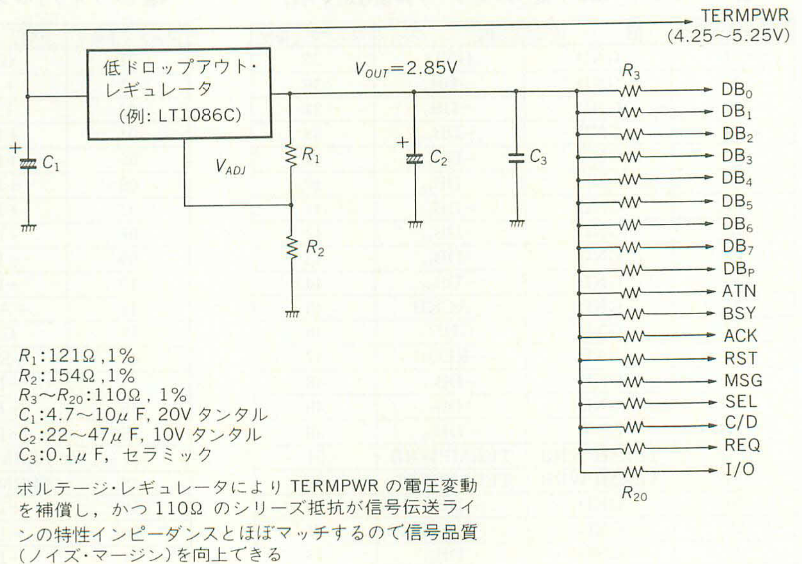
コンタクト番号	信号	信号	コンタクト番号
01	GND	GND	35
02	GND	$-DB_8$	36
03	GND	$-DB_9$	37
04	GND	$-DB_{10}$	38
05	GND	$-DB_{11}$	39
06	GND	$-DB_{12}$	40
07	GND	$-DB_{13}$	41
08	GND	$-DB_{14}$	42
09	GND	$-DB_{15}$	43
10	GND	$-DB_{p1}$	44
11	GND	−ACKB	45
12	GND	GND	46
13	GND	−REQB	47
14	GND	$-DB_{16}$	48
15	GND	$-DB_{17}$	49
16	GND	$-DB_{18}$	50
17	TERMPWRB	TERMPWRB	51
18	TERMPWRB	TERMPWRB	52
19	GND	$-DB_{19}$	53
20	GND	$-DB_{20}$	54
21	GND	$-DB_{21}$	55
22	GND	$-DB_{22}$	56
23	GND	$-DB_{23}$	57
24	GND	$-DB_{p2}$	58
25	GND	$-DB_{24}$	59
26	GND	$-DB_{25}$	60
27	GND	$-DB_{26}$	61
28	GND	$-DB_{27}$	62
29	GND	$-DB_{28}$	63
30	GND	$-DB_{29}$	64
31	GND	$-DB_{30}$	65
32	GND	$-DB_{31}$	66
33	GND	$-DB_{p3}$	67
34	GND	GND	68

〈表 6〉ディファレンシャル型バス-B ケーブル信号割り付け

コンタクト番号	信号	信号	コンタクト番号
01	GND	GND	35
02	$+DB_8$	$-DB_8$	36
03	$+DB_9$	$-DB_9$	37
04	$+DB_{10}$	$-DB_{10}$	38
05	$+DB_{11}$	$-DB_{11}$	39
06	$+DB_{12}$	$-DB_{12}$	40
07	$+DB_{13}$	$-DB_{13}$	41
08	$+DB_{14}$	$-DB_{14}$	42
09	$+DB_{15}$	$-DB_{15}$	43
10	$+DB_{p1}$	$-DB_{p1}$	44
11	+ACKB	−ACKB	45
12	GND	DIFFSENS	46
13	+REQB	−REQB	47
14	$+DB_{16}$	$-DB_{16}$	48
15	$+DB_{17}$	$-DB_{17}$	49
16	$+DB_{18}$	$-DB_{18}$	50
17	TERMPWRB	TERMPWRB	51
18	TERMPWRB	TERMPWRB	52
19	$+DB_{19}$	$-DB_{19}$	53
20	$+DB_{20}$	$-DB_{20}$	54
21	$+DB_{21}$	$-DB_{21}$	55
22	$+DB_{22}$	$-DB_{22}$	56
23	$+DB_{23}$	$-DB_{23}$	57
24	$+DB_{p2}$	$-DB_{p2}$	58
25	$+DB_{24}$	$-DB_{24}$	59
26	$+DB_{25}$	$-DB_{25}$	60
27	$+DB_{26}$	$-DB_{26}$	61
28	$+DB_{27}$	$-DB_{27}$	62
29	$+DB_{28}$	$-DB_{28}$	63
30	$+DB_{29}$	$-DB_{29}$	64
31	$+DB_{30}$	$-DB_{30}$	65
32	$+DB_{31}$	$-DB_{31}$	66
33	$+DB_{p3}$	$-DB_{p3}$	67
34	GND	GND	68

〈表7〉SCSIバスの電気的条件

				SCSI2	SCSI1
シングル・エンド型バス(最大ケーブル長6m)	ドライバ特性	"L"レベル出力電圧：V_{OL}		0.0～0.5VDC	0.0～0.4VDC
		"H"レベル出力電圧：V_{OH}		2.5～5.25VDC	2.5～5.25VDC
		"L"レベル出力電流：I_{OL}		48mA (@0.5VDC)	48mA(@0.5VDC)
	レシーバ特性	"L"レベル入力電圧：V_{IL}		0.0～0.8VDC	0.0～0.8VDC
		"H"レベル入力電圧：V_{IH}		2.0～5.25VDC	2.0～5.25VDC
		"L"レベル入力電流：I_{IL}		−0.4～0.0mA(@0.5VDC)	−0.4mA(@0.4VDC)
		"H"レベル入力電流：I_{IH}		0.0～0.1mA(@2.7VDC)	―――
		ヒステリシス：ΔV_{TH}		0.2VDC min.	0.2VDC min.
		入力容量		25pF max.	―――
	ターミネータ用電源	供給電圧：V_{TERM}		4.25～5.25VDC	4.0～5.25VDC
		電流容量	Aケーブル	900mA min.	800mA min.
			Bケーブル	1500mA min.	―――
ディファレンシャル型バス(最大長25m)	ドライバ特性	"L"レベル出力電圧　：V_{OL}		1.7VDC max.(I_{OL}=55mA)	2.0VDC max.(I_{OL}=55mA)
		"H"レベル出力電圧　：V_{OH}		2.7VDC min.(I_{OH}=−55mA)	3.0VDCmin.(I_{OH}=−55mA)
		ディファレシャル電圧　：V_{OD} (コモン・モード電圧：−7～+12VDC)		1.0VDC min.	1.0VDC min.
	レシーバ特性	入力電流　：I_I (入力電圧：−7～+12VDC)		±2.0mA max.	±2.0mA max.
		ヒステリシス　：ΔV_{TH}		35mV min.	35mV min.
		入力容量		25pF max.	―――
	ターミネータ用電源	供給電圧　：V_{TERM}		4.0～5.25VDC	4.0～5.25VDC
		電流容量	Aケーブル	600mA min.	600mA min.
			Bケーブル	1000mA min.	―――



〈図6〉シングル・エンド型バス用のアクティブ・ターミネータ

また，イニシエータに対してはメッセージの種別ごとにアテンション・コンディションを継続してもよいか否かの条件が規定されました(**表9**，**表10**)．

● FAST SCSI

FAST SCSIは同期モードのデータ転送フェーズでのみ使用できます．SDTR(SYNCHRONOUS DATA TRANSFER REQUEST)メッセージで同期モード転送のパラメータをSCSIデバイス間で定義するとき，転送周期(Transfer Period)が200nsより小さい値，すなわち5Mバイト/sを超える転送速度に決定された場合にはFAST SCSIのタイミング規約を適用してデータ転送を実行します．

なお，Transfer Periodの最小値は100ns(=10Mバイト/s)です．SDTRメッセージで200ns以上のTransfer Period値が決定されたときにはSCSI1のタイミング規定で動作しなければなりません．

図7にデータ転送のタイミング規定を示します．FAST SCSIの転送タイミングは相当に厳しい値なの

〈表8〉タイミング規約

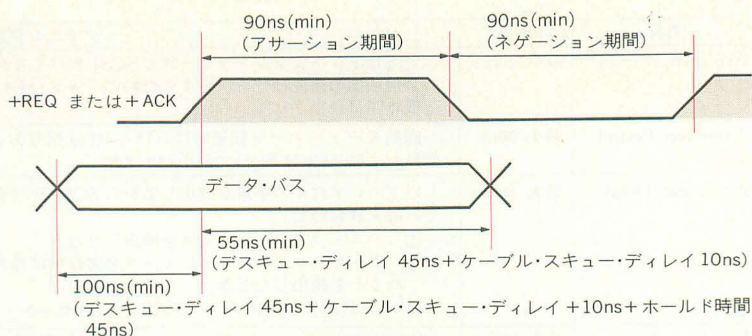
名称	規定値	タイミング規定
Arbitration Delay	最小 2.4μs	アービトレーション・フェーズでSCSIデバイスがBSY信号およびSCSI IDを送出してからバス使用権の優先順位を決定するためにデータ・バス上の値を判定するまでの最小待機時間．最大待機時間は規定されない
Assertion Period	最小 90ns	同期モードのデータ転送時におけるREQ信号およびACK信号(Bケーブルついては REQB信号およびACKB信号)の最小パルス幅
Bus Clear Delay	最大 800ns	以下のいずれかの事象が発生してからSCSIデバイスがすべてのバス信号の駆動を停止するまでの最大許容時間． ① バス・フリー・フェーズを検出したとき ② アービトレーション・フェーズを実行中に他のSCSIデバイスがSEL信号をTRUEにしたことを検出したとき ③ RST信号がTRUEになったこと(リセット・コンディション)を検出したとき
Bus Free Delay	最小 800ns	SCSIデバイスがバス・フリー・フェーズを検出してからアービトレーション・フェーズを開始するためにBSY信号およびSCSI IDを送出するまでの最小待機時間
Bus Set Delay	最大 1.8μs	SCSIデバイスがバス・フリー・フェーズを検出してからアービトレーション・フェーズを開始するためにBSY信号とSCSI IDとを送出するまでの最大許容時間
Bus Settle Delay	最小 400ns	特定の制御信号の状態が変化してからバスの状態が安定するまでの最小待機時間
Cable Skew Delay	最大 10ns	任意の2台のSCSIデバイス間で任意のバス信号間のインターフェース・ケーブル上の信号伝播時間の最大許容差
Data Release Delay	最大 400ns	I/O信号の状態がFALSEからTRUEに変化した後，イニシエータがデータ・バス信号の駆動を停止するまでの最大許容時間
Deskew Delay	最小 45ns	バス信号間の伝送スキュー補償の最小時間
Disconnection Delay	最小 200μs	ターゲットがイニシエータからのDISCONNECTメッセージによりバス・フリー・フェーズに移行した後，つぎにアービトレーションに参加するまでの最小待機時間
Hold Time	最小 45ns	同期モードのデータ転送においてデータを受信するSCSIデバイスでのホールド時間を補償するためにREQ信号またはACK信号(Bケーブルについては，REQB信号とACKB信号)パルスの前縁からデータ・バス上の転送データを保持していなければならない最小時間
Negation Period	最小 90ns	同期モードのデータ転送においてREQ信号の後縁から次のREQ信号の前縁まで，またはACK信号の後縁からつぎのACK信号の前縁まで(BケーブルではREQB信号とACKB信号)の最小時間
Power-On to Selection Time	最大 10s [推奨値]	電源投入後にターゲットがTEST UNIT READY，INQUIRY，またはREQUEST SENSEコマンドに応答できるようになるまでの最大許容時間
Reset Hold Time	最小 25μs	リセット・コンディションを生成するときRST信号をTRUEにする最小保持時間．最大時間は規定されない
Reset to Selection Time	最大 10s [推奨値]	ハード・リセット・コンディションが解除された後にターゲットがTEST UNIT READY，INQUIRY，またはREQUEST SENSEコマンドに応答できるようになるまでの最大許容時間
Selection Abort Time	最大 200μs	セレクション・フェーズまたはリセレクション・フェーズでSCSIデバイスが自己が選択されていることを認識してから，BSY信号を応答するまでの最大許容時間
Selection Timeout Delay	最小 250ms [推奨値]	セレクション・フェーズまたはリセレクション・フェーズでイニシエータまたはターゲットがタイムアウト処理を開始するまでに選択対象のSCSIデバイスからのBSY信号応答を待つ最小時間
Transfer Period	100ns-1020ns	同期モードのデータ転送においてREQ信号の前縁から次のREQ信号の前縁まで，またはACK信号の前縁からつぎのACK信号の前縁まで(BケーブルではREQB信号とACKB信号)の最小時間〔最小繰り返し時間〕．具体的な値はSDTRメッセージの交換により，イニシエータとターゲット間で決定される
Fast Assertion Period	最小 30ns	FAST同期モードのデータ転送時におけるREQ信号およびACK信号(BケーブルではREQB信号およびACKB信号)の最小パルス幅
Fast Cable Skew Delay	最大 5ns	FAST同期モード転送を行うときの任意の2台のSCSIデバイス間での任意のバス信号間のインターフェース・ケーブル上の信号伝播時間の最大許容差
Fast Deskew Delay	最小 20ns	FAST同期モード転送を行うときのバス信号間の伝送スキュー補償の最小時間
Fast Hold Time	最小 10ns	FAST同期モードのデータ転送においてデータを受信するSCSIデバイスでのホールド時間を補償するため，REQ信号またはACK信号(Bケーブルについては REQB信号とACKB信号)パルスの前縁からデータ・バス上の転送データを保持していなければならない最小時間
Fast Negation Period	最小 30ns	FAST同期モードのデータ転送においてREQ信号の後縁から次のREQ信号の前縁まで，またはACK信号の後縁からつぎのACK信号の前縁まで(BケーブルではREQB信号とACKB信号)の最小時間

で，適用に際してはドライバ/レシーバやケーブルの特性に十分注意する必要があります．特にシングル・エンド型バスではケーブル長やシステム構成およびデータ転送速度の上限を制限するなどの考慮が必要になると思われます．SCSI2の規格書にはシングル・エンド型バスではタイミング・パラメータのテストがされていないこと，およびFAST SCSIは推奨できないことが注記されています．

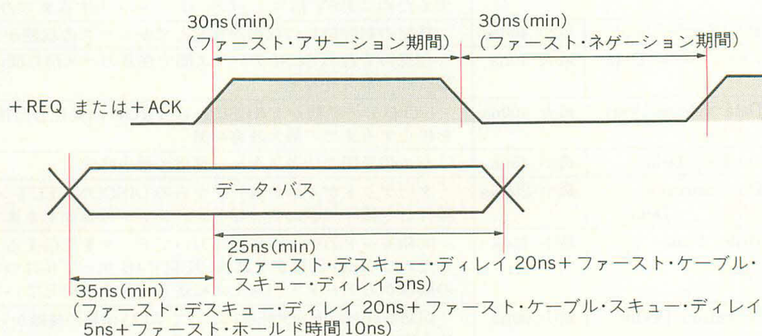
WIDE SCSI

WIDE SCSIは拡張データ・バス(Bケーブル)を使用してAケーブル上のデータ・バスと並列転送をすることにより，16ビット幅または32ビット幅でのデー

〈図7〉
同期モードのデータ転送タイミング規定
(送端側デバイス)



(a) 5Mバイト/s以下の時のタイミング：SCSI1と同一



(b) 5Mバイト/sを超える時のタイミング：FAST SCSI

タ転送をすることができます．使用するバス幅は16ビットまたは32ビットのどちらかを選択することができますが，32ビット転送をサポートするときは16ビットでの転送も動作できるようにしておくことが推奨されています．WIDE転送はデータ・フェーズでのみ使用可能であり，コマンドなどの他の情報転送はAケーブルの8ビットのデータ・バスで実行します．

WIDE転送を使用するときは，そのSCSIデバイス間でWDTR(WIDE DATA TRANSFER REQUEST)メッセージを交換してデータ転送時に使用するデータ・バスの幅を定義しておきます．メッセージ交換の手順は同期モード転送を行うときのSDTRメッセージの交換手順と同様です．WIDE転送では一度に2バイトまたは4バイトの転送を行いますが，このときのデータ・バイトの転送順番を図8に示します．

WIDE転送は非同期モードでも同期モードでも実行することができますが，同期モード転送のパラメータ設定はWDTRメッセージによりクリアされることになっているため，同期モードのWIDE転送を行う場合はWDTRメッセージの交換の後にSDTRメッセージの交換を実行する必要があります．

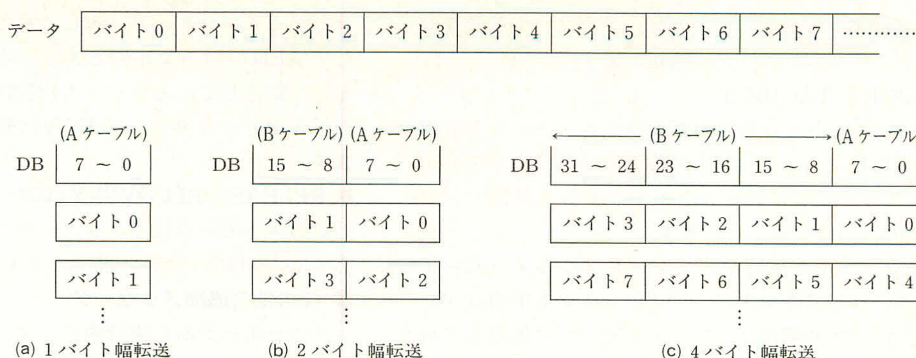
Aケーブルと Bケーブルでは同一のモード(非同期モードまたは同一転送パラメータでの同期モード)でデータ転送を行いますが，転送タイミングはそれぞれに個別の制御線(Aケーブル：REQ/ACK，Bケーブル：REQB/ACKB)で制御されます．したがって，両者のケーブル長などの相違についての制約がなくなり，図9に示すようにBケーブルを持った装置とAケーブルだけの装置を混在して接続することもできます．

2本のケーブル間での転送タイミングの規定はありませんが，ターゲットはREQとREQBの発行タイミングを，イニシエータはACKとACKBの応答タイミングを制御することで，A/Bケーブルの転送を同期させることができます．データ転送の正常性を保証するため，ターゲットはひとつのデータ・フェーズでのREQ/ACKハンドシェイク数とREQB/ACKBハンドシェイク数が等しいことを確認します．

転送データのバイト数が2または4の倍数でないために転送の最後のデータがバス幅に満たないときは残りのバイト位置にはどんな値を送出しても構いませんが，パリティの値は正しくなければなりません．データ・イン・フェーズでバス幅に満たない転送をしたときはターゲットはIGNORE WIDE RESIDUEメッセージをデータ・フェーズの直後に送信して無効なデータ・バイト数をイニシエータに通知します．

● バス・フェーズのシーケンス

アービトレーション・フェーズとリセレクション・フェーズ，およびセレクション・フェーズでのアテンション・コンディションが標準機能になったので，バ



〈図8〉
WIDE SCSIでのデータ転送順序

ス・フェーズの推移シーケンスは図10に示すようになります.

メッセージ・システム

メッセージ・システムはインターフェースの物理レイヤの管理を目的としていましたが，SCSI2ではコマンド・キューイングなどの論理階層も制御するように仕様が拡張され，新たに2バイト長のメッセージなどが追加されました．表9と表10にSCSI2で定義されているメッセージの一覧を示します．メッセージ・プロトコルの曖昧さを避けるため，イニシエータが送信するいくつかのメッセージについては，そのメッセージの送信終了時にアテンション・コンディションを解除しなければならないことが規定されました.

● 機能が拡張されたメッセージ

▶ DISCONNECT(04h)

イニシエータからもこのメッセージを発行できるようになりました．従来ディスコネクト処理はターゲットが決定するタイミングでのみ可能でしたが，この機能拡張によりイニシエータはアテンション・コンディションを生成してターゲットにディスコネクトを要求することができます.

なお，このメッセージの指示でディスコネクトを実行したターゲットは200μs(Disconnection Delay)を経過するまでつぎのアービトレーションを開始できません.

▶ IDENTIFY(80h~FFh)

ターゲット・ルーチンと呼ばれる機能が規定され，このメッセージは図11に示すような定義になりました．ターゲット・ルーチンについては保守や診断を目的としてターゲット上(ロジカル・ユニットではない)で実行される機能で，INQUIRYあるいはREQUEST SENSEコマンドとのみ組み合わせて使用します．最大8つまでのターゲット・ルーチンを定義できるとされていますが，具体例は明らかではありません．IDENTIFYメッセージでLUNTARビットに“1”を指定することによってターゲット・ルーチンを起動します．この場合，LUNTRNフィールドはロジカル・ユニット番号ではなく，ターゲット・ルーチンの種別を指定することになります.

● コマンド・キューイング用のメッセージ

タグ付(Tagged)コマンド・キューイングの仕組みについては後で詳細に説明しますが，以下の5種類のメッセージをサポートする必要があります.

最初の3つのメッセージ(キュー・タグ・メッセージと呼ぶ)は図12に示す構造を持つ2バイト長のメッセージで，イニシエータが発行する個々のコマンドに固有な識別子(キュー・タグ)を与えるためのものです．キュー・タグは符号なしの2進数で，最大256コマンドまでの識別が可能です.

▶ HEAD OF QUEUE TAG(21h)

このメッセージで発行されたコマンドはLIFO(Last-In, First-Out)の実行順序となるようにコマンド・キューの先頭に置かれ，実行中のコマンドが終了すると最優先で実行されます.

▶ ORDERED QUEUE TAG(22h)

このメッセージはコマンドを到着順に実行することを指定します．このメッセージで発行されたコマンドは，すでにキューイングされている他のすべてのコマンドが終了した後に実行され，また，そのコマンドの後に到着したコマンドは，HEAD OF QUEUE TAGメッセージで発行されたコマンドを除いてそのコマンドの後に実行されます.

▶ SIMPLE QUEUE TAG(20h)

このメッセージで発行されたコマンドについてはターゲットが実行順序を変更することができます．一般にターゲットはキューイングされているコマンド列を性能が最適化されるような順序で実行しますが，実行順序の変更範囲を制約できるようにするためのモード・セレクト・パラメータも用意されています.

また，ターゲットはタグ付コマンドのリコネクショ

ンを実行するときにこのメッセージでコマンドのキュー・タグをイニシエータに通知します．

▶ ABORT TAG(0Dh)

このメッセージは指定したロジカル・ユニットで実行中またはキューイング中の特定のひとつのコマンド(このメッセージを発行したイニシエータが発行したコマンド)のみをクリアします．

SCSIバス上で実行中のコマンドはこのメッセージだけでクリアできますが，ディスコネクト中のコマンドをクリアする場合にはこのメッセージに先立ってロジカル・ユニット番号(IDENTIFY)とクリアするコマンドのキュー・タグ(SIMPLE QUEUE TAG)を指定します．

▶ CLEAR QUEUE(0Eh)

このメッセージは指定したロジカル・ユニットで実行中またはキューイング中のすべてのコマンド(どのイニシエータが発行したコマンドかにかかわらず)をクリアします．

このメッセージを発行したイニシエータ以外のイニシエータからのコマンドがクリアされた場合にはそれらのイニシエータに対してユニット・アテンション・コンディションが発生します．

● WIDE SCSI用のメッセージ

WIDE SCSIのために必要なメッセージとしてつぎのふたつがあります．

▶ WDTR(拡張メッセージ：03h)

イニシエータとターゲット間でこのメッセージを交換し，データ転送に使用するバスの幅を決定するものです(図13)．

▶ IGNORE WIDE RESIDUE (23h)

データ・イン・フェーズの最後のバイト数がバス幅に満たないとき，ターゲットからイニシエータに対して無効なバイト数を通知するために使用します(図14)．

● ECA(Extended Contingent Allegiance)用のメッセージ

ECA状態の規定については後で説明しますが，その制御用としてつぎのふたつのメッセージがあります．

▶ INITIATE RECOVERY(0Fh)

このメッセージはECA状態の開始を宣言するもので，通常はターゲットからイニシエータに対して発行されます．例外としてAENプロトコルのときに一時的にイニシエータとして動作しているデバイス(通常はターゲット)がこのメッセージを発行してECA状態に入ることができます．

▶ RELEASE RECOVERY(10h)

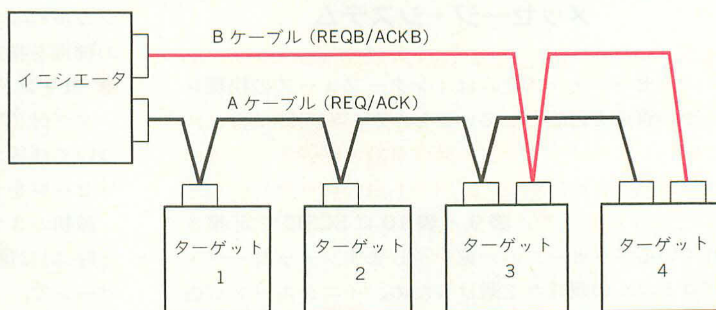
このメッセージはイニシエータからターゲットに対してECA状態の解除を指示するものです．

● その他の追加メッセージ

イニシエータから実行中のコマンドを途中で終了させる(それまでの実行結果は保証して終了させる)手段としてTERMINATE I/O PROCESSメッセージが用意されました．

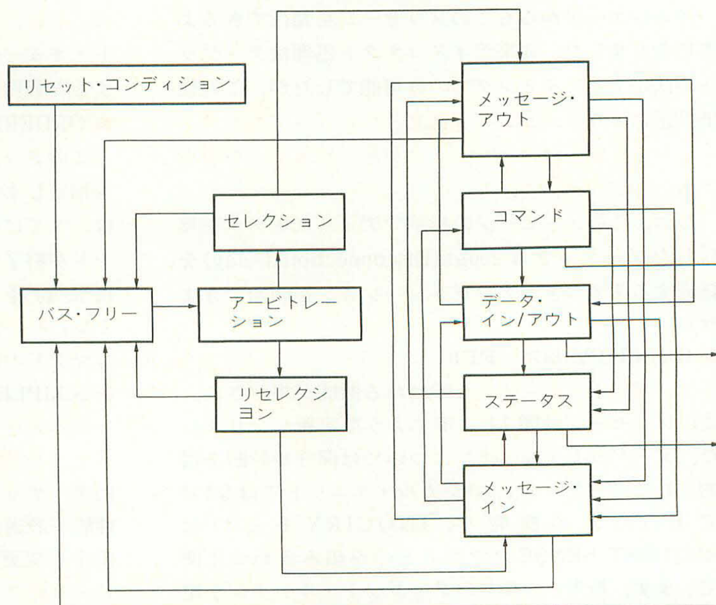
このメッセージを受信したターゲットは可能な限り

〈図9〉Aケーブルと A/B ケーブル(WIDE SCSI)の混在接続



Aケーブルと Bケーブルのデータ転送はそれぞれに専用の制御線(REQ/ACK, REQB/ACKB)でハンドシェイクされるのでAケーブルだけの装置とA/Bケーブルをもった装置を混在して接続できる．ケーブル長は異なってもよい

〈図10〉バス・フェーズの推移シーケンス



〈表 9〉メッセージ一覧

メッセージ・コード	メッセージ名称	ATN 解除条件	方向	サポートの要否			
				SCSI2		CCS	SCSI1
				I	T		
00h	COMMAND COMPLETE	—	IN	M	M	M	M
01h	拡張メッセージ(表 10)	—	—	—	—	—	—
02h	SAVE DATA POINTER	—	IN	O	O	O	O
03h	RESTORE POINTERS	—	IN	O	O	O	O
04h	DISCONNECT	—	IN	O	O	O	O
		YES	OUT	O	O	—	—
05h	INITIATOR DETECTED ERROR	YES	OUT	M	M	O	O
06h	ABORT	YES	OUT	0	M	M	O
07h	MESSAGE REJECT	—	IN	M	M	M	O
		YES	OUT	M	M	M	O
08h	NO OPERATION	YES	OUT	M	M	M	O
09h	MESSAGE PARITY ERROR	YES	OUT	M	M	O	O
0Ah	LINKED COMMAND COMPLETE	—	IN	O	O	O	O
0Bh	LINKED COMMAND COMPLETE WITH FLAG	—	IN	O	O	O	O
0Ch	BUS DEVICE RESET	YES	OUT	O	M	M	O
0Dh	ABORT TAG	YES	OUT	O	O	R	R
0Eh	CLEAR QUEUE	YES	OUT	O	O	R	R
0Fh	INITIATE RECOVERY	—	IN	O	O	R	R
		YES	OUT	O	O	R	R
10h	RELEASE RECOVERY	YES	OUT	O	O	R	R
11h	TERMINATE I/O PROCESS	YES	OUT	O	O	R	R
12h～1Fh	(1 バイト長メッセージ)	—	—	R	R	R	R
20h	SIMPLE QUEUE TAG	—	IN	O	O	R	R
		NO	OUT	O	O	R	R
21h	HEAD OF QUEUE TAG	NO	OUT	O	O	R	R
22h	ORDERED QUEUE TAG	NO	OUT	O	O	R	R
23h	IGNORE WIDE RESIDUE	—	IN	O	O	R	R
24h～2Fh	(2 バイト長メッセージ)	—	—	R	R	R	R
30h～7Fh		—	—	R	R	R	R
80h～FFh	IDENTIFY	—	IN	M	O	O	O
		NO	OUT	M	M	O	M(T)

I：イニシエータ，T：ターゲット，
M：サポート必須，O：オプション，R：リザーブ
IN：ターゲット→イニシエータ，OUT：イニシエータ→ターゲット

表中の ATN 解除条件の欄が“YES”のメッセージをイニシエータが送信するとき，イニシエータはそのメッセージ・アウト・フェーズの最後の ACK 信号応答の 90ns(Deskew Delay×2)以上前に ATN 信号を FALSE にし，アテンション・コンディションを解除しなければならない．この条件に違反したとき，ターゲットはプロトコル上のエラーとみなしてバス・フリー・フェーズに移行する

〈表 10〉拡張メッセージ一覧

拡張メッセージ・コード	メッセージ長(バイト)	メッセージ名称	ATN 解除条件	方向	サポートの要否			
					SCSI2		CCS	SCSI1
					I	T		
00h	7	MODIFY DATA POINTER	—	IN	O	O	O	O
01h	5	SYNCHRONOUS DATA TRANSFER REQUEST	—	IN	O	O	O	O
			YES	OUT	O	O	O	O
02h	—	(EXTENDED IDENTFY)⇨削除	—	—	R	R	O	O
03h	4	WIDE DATA TRANSFER REQUEST	—	IN	O	O	O	O
			YES	OUT	O	O	O	O
04h～7Fh	—		—	—	R	R	R	R
80h～FFh	—		—	—	V	V	V	V

I：イニシエータ，T：ターゲット，
O：オプション，V：ベンダ固有，R：リザーブ
IN：ターゲット→イニシエータ，OUT：イニシエータ→ターゲット
ATN 解除条件：表 9

〈図11〉
IDENTIFY メッセージ

ビット	7	6	5	4	3	2	1	0
バイト0	1	DiscPriv	LUNTAR	0	0	LUNTRN		

LUNTAR	LUNTRN
0	ロジカル・ユニット番号(LUN)
1	ターゲット・ルーチン番号

〈図12〉
キュー・タグ・メッセージ

ビット	7	6	5	4	3	2	1	0
バイト0	メッセージ・コード：20h，21h，または22h							
バイト1	キュー・タグ：00h～FFh							

キュー・タグ・メッセージはアイデンティファイ・メッセージに続けて発行され，コマンドの識別子(キュー・タグ)を宣言する

早い時点でコマンドの実行を打ち切り，特別なステータス(COMMAND TERMINATED)でそのコマンドを終了させます．このときターゲットはセンス・データを生成し，そのコマンドがデータ転送をともなうものであれば実行停止時までに正常に転送を完了したデータ量をイニシエータが算出できるような情報をセンス・データに表示します．

その他にANSIではディスク・アレイなどのアプリケーションを意図して，ターゲットとのデータ転送の開始タイミングをイニシエータから制御するためのメッセージを検討していますが，SCSI2の最終規格に組み込まれるか否かは明らかではありません．

● 削除されたメッセージ

ロジカル・ユニット番号(LUN)を拡張するために定義されていたEXTENDED IDENTIFYメッセージ(拡張メッセージ・コード：02h)が削除されました．

共通論理仕様

すべてのデバイス・タイプに共通な論理仕様で，規定が変更されたり機能が追加された主要な項目を以下に説明します．

● コマンド(CDB：Command Descripter Blok)の記述規定

▶ グループ2のCDB

10バイト長のCDBとしてグループ2(オペレーション・コード：40h～5Fh)が使用されるようになりました．

▶ LUNフィールド

ロジカル・ユニット番号(LUN)は，必須機能となったINENTIFYメッセージで指定されるのでCDB上のLUNフィールドは意味を持たなくなりましたが，SCSI1との整合性を保つため従来どおりに定義が残されました．ターゲットはこのフィールドに指定された値は無視します．なお，このフィールドは将来の規格拡張で別の機能に使用される可能性があるので，ゼロを指定しておくことが推奨されています．

● ステータス・バイト

図15にステータス・バイトの定義を示します．ビット6，5，0はSCSI1ではベンダ固有に使用できましたが，それが許されなくなりました．

また，機能の追加にともなって以下のステータス・コードが追加されました．

▶ COMMAND TERMINATED

このステータスはTERMINATE I/Oメッセージを受け取ったことによってコマンドの実行が終了したことを示します．

▶ QUEUE FULL

このステータスはコマンド・キューに空きがなく，新たなタグ付コマンドを受け取ってキューイングできないことを示します．

● プログラマブル動作モード

CHANGE DEFINITIONコマンド(オプション)を用いて，ターゲットあるいはロジカル・ユニットの動作モード(SCSIの論理仕様レベル)をイニシエータが指定できるようになりました．イニシエータはSCSI1，CCS，またはSCSI2のいずれかを指定することができます．

ターゲットはコマンド・セットやパラメータの定義，コマンドの動作などを指定されたSCSIレベルに変更しますが，その他のパラメータ(ベンダID，デバイス・モデル名など)をベンダ固有の規定で変更することも許されています．

イニシエータはINQUIRYコマンドで，ターゲットとロジカル・ユニットの現在の動作モードやサポートされている機能レベルを知ることができます．

この機能は特定のSCSI仕様に依存しているアプリケーションをSCSI2の環境下で共存させることが必要なときに有用な手段になります．

● AEN(非同期事象の通知)機構

ロジカル・ユニットで非同期に発生した事象をターゲットからイニシエータに通知するための機能として

〈図 13〉
WDTR(WIDE DATA TRANSFER REQUEST)メッセージ

ビット	7	6	5	4	3	2	1	0
バイト0	メッセージ・コード：01h(拡張メッセージ)							
バイト1	拡張メッセージ長：02h							
バイト2	拡張メッセージ・コード：03h							
バイト3	転送バイト幅：$m(2^m)$							

バイト3の値がデータ転送時のデータ・バス幅を示す
m=00h： 8ビット幅
01h：16ビット幅
02h：32ビット幅

〈図 14〉
IGNORE WIDE RESIDUE メッセージ

ビット	7	6	5	4	3	2	1	0
バイト0	メッセージ・コード：23h							
バイト1	無効データ・バイト数							

バイト1の値はデータ・イン・フェーズの最後のREQ(B)/ACK(B)ハンドシェイクでの無効な転送バイト数を示す

● 無効なデータ・バイト

バイト1の値	32ビット転送	16ビット転送
00h	——	——
01h	DB_{31}～DB_{24}	DB_{15}～DB_{8}
02h	DB_{31}～DB_{16}	——
03h	DB_{31}～DB_{8}	——

AEN(Asynchronous Event Notification)機構が設けられました．

SCSI1ではこの機能がなかったので，ロジカル・ユニットの状態遷移などを検出するにはイニシエータからのポーリングが必要でした．

代表的な非同期事象にはつぎのものがありますが，ベンダ固有の追加定義もできます．

・I/O装置の状態遷移(ノット・レディ→レディ，媒体交換やオペレータ操作など)
・I/O動作の終了(MT装置のリワインドなど)
・正常終了報告済みのコマンドでのエラー(ライト・バック・キャッシュなど)
・ユニット・アテンション・コンディション

AENを使用するターゲットにはイニシエータとして動作できる機能が必要です．

また，AENの通知先はプロセッサ・デバイスに限定されています．

つまり，AENを使用するイニシエータは，一時的にターゲット(プロセッサ・デバイス)として動作できなければなりません．

AEN機能を備えたターゲットは初期化(電源投入時など)時に以下の手順でどのSCSIデバイスがAEN受信機能を持っているかを調べ，そのSCSIデバイスに対してのみAENを実行します．

① 自分以外のすべてのSCSI IDについてセレクション・フェーズを実行して，応答したSCSIデバイスに対し，LUN=0を指定してINQUIRYコマンドを発行する．

② INQUIRYデータ上のデバイス・タイプがプロセッサ・デバイスで，かつAENの受信が可能であること(AENCビット=1)が表示されていればテスト・ユニット・レディ・コマンドを発行する．

③ 必要に応じてREQUEST SENSE，TEST UNIT READYを繰り返し，TEST UNIT READYコマンドがGOODステータスで終了すればそのSCSIデバイスはAENを受信できるものと判断する．

AENの事象が発生したとき，そのターゲットは一時的にイニシエータとして動作し，送信先のSCSIデバイスのLUN=0を選択してSENDコマンドを発行します．このSENDコマンドでセンス・データと同一形式のデータを転送することにより，発生した非同期事象の詳細を通知します(図16)．

AENの事象がECA状態の生成を必要とするときはSENDコマンドのコマンド・フェーズの前にINITIATE RECOVERYメッセージを送信することでECA状態に入ることができます．

● CA(Contingent Allegiance)状態

CA状態はエラー・リカバリのための情報であるセンス・データの保持条件を明確にしたものです．

CA状態はターゲットがCHECK CONDITIONステータスまたはCOMMAND TERMINATEDステータスを報告したとき，あるいはターゲットが強制的にバス・フリー・フェーズに移行(SCSIバス・プロトコル上の重度なエラーを検出したとき)してセンス・データを生成したとき，そのときのイニシエータ-ターゲット-ロジカル・ユニット(またはターゲット・ルーチン)のパスに対して発生し，同一のパスでつぎのコマ

〈図 15〉ステータス・バイト

ビット	7	6	5	4	3	2	1	0
バイト0	R	R	ステータス・バイト・コード					R

R：リザーブ（=0）

ビット6,5,0はベンダ固有に定義できなくなった

ビット 54321	
00000	GOOD
00001	CHECK CONDITION
00010	CONDITION MET
00100	BUSY
01000	INTERMEDIATE
01010	INTERMEDIATE – CONDITION MET
01100	RESERVATION CONFLICT
10001	COMMAND TEAMNATED
10100	QUEUE FULL

ンドが発行されるかハード・リセット・コンディション，ABORTメッセージ，またはBUS DEVICE RESETメッセージによって解除されます．

通常は，イニシエータがREQUEST SENSEコマンドを発行してセンス・データを読み出します．

CA状態のとき，ターゲットはそのパスに対してのセンス・データを保持し続けます．ターゲットが他のパスに対して個別にセンス・データを保持できないのであれば，そのターゲットはCA状態が解除されるまで他のパスのアクセスに対してBUSYステータスを応答します(図17)．

● ECA(Extended Contingent Allegience)状態

CA状態がセンス・データの排他的保持だけの機能であるのに対し，ECA状態はエラー・リカバリ処理のための完全な排他制御の手順を提供します．

この機能によってエラー終了したコマンドの後処理やキューイングされているコマンドの処置などの一連のエラー・リカバリ処理を行う間，イニシエータはそのロジカル・ユニットを排他的に占有することができます(図18)．

ECA状態はCHECK CONDITIONステータスやCOMMAND TERMINATEDステータスの直後(COMMAND COMPLETEメッセージの前)あるいはAENプロトコルのSENDコマンドの前にターゲット(AENのときはイニシエータとして動作している)がINITIAE RECOVERYメッセージを発行することで発生し，そのイニシエータがRELEASE RECOVERYメッセージを発行するか，ハード・リセット・コンディションまたはBUS DEVICE RESETメッセージが発行されるまで継続します．ECA状態にあるターゲットは，そのロジカル・ユニットに対する他のイニシエータからのアクセスに対してはBUSYステータスを応答します．

ECA状態はターゲットの宣言によってのみ生成されます．CA状態とECA状態のどちらを発生させるかは発生したエラー状態などによってターゲットが決めることになりますが，その基準はSCSI2規格の中には規定されていません．

● コマンド・キューイング

コマンド・キューイングにはタグなし(Untagged)キューイングとタグ付き(Tagged)キューイングの2種類の方法がありますが，SCSI2で“コマンド・キューイング”というときには一般にタグ付きキューイングを指します．

ターゲットは両方のキューイング機能をサポートできます．ひとつのイニシエータはある一時点ではいずれか一方のキューイング手法のみを使用できますが，マルチ・イニシエータのときは他のイニシエータは別のキューイング手法を使用してもよいのでターゲットは両方のキューイングの混在制御が必要になることもあります．

▶タグなしキューイング

ターゲットはロジカル・ユニット上で実行中のコマンドがあっても他のイニシエータからそのロジカル・ユニットに発行されたコマンドを受け付けてキューイングします．

キューイング可能なコマンドは各ロジカル・ユニットについてイニシエータあたりひとつまでです．

イニシエータはあるロジカル・ユニットに対して発行したコマンドが終了するまでは同一のロジカル・ユニットに別のコマンドを発行することはできません．このキューイング手法はBUSYステータスによるコマンド再発行のオーバ・ヘッド時間，およびターゲットがつぎのコマンドを開始するまでのオーバ・ヘッド時間などを削減させることができますが，マルチイニシエータのときにしか効果がありません．

▶タグ付きキューイング

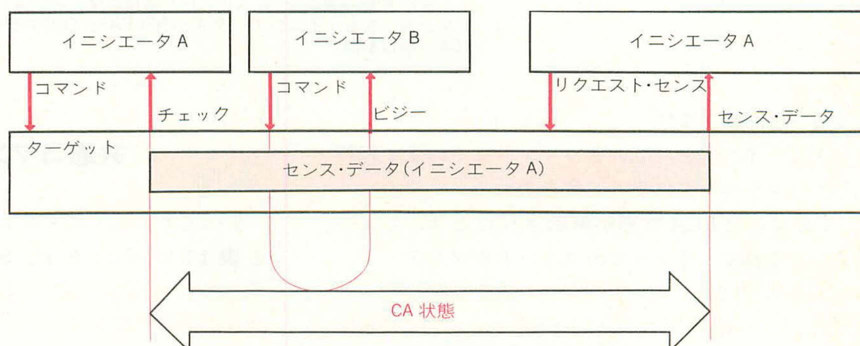
ターゲットは各ロジカル・ユニットについて同一ま

〈図 16〉
AEN データ形式
(SEND コマンド)

ビット	7	6	5	4	3	2	1	0
バイト 0	(リザーブ：00)		LUNTAR	(リザーブ：00)		LUNTRN		
1	(リザーブ：00h)							
2	(リザーブ：00h)							
3	(リザーブ：00h)							
4	センス・データ・バイト(0)							
⋮	⋮ (図 20 と同一形式) 非同期事象の詳細を表す							
$n+4$	センス・データ・バイト(n)							

LUNTAR	LUNTRN	非同期事象の発生元
0	LUN 番号	ロジカル・ユニット
1	ターゲット・ルーチン番号	ターゲット・ルーチン

〈図 17〉CA 状態



イニシエータ A に対してセンス・データが生成されると CA 状態が発生する．ターゲットが他のイニシエータ用に個別にセンス・データを保持できないのであれば，CA 状態の時にイニシエータ B からコマンドを受け取った時はBUSY・ステータスを応答する．ターゲットは CA 状態の間，他のコマンドを実行しないのでイニシエータ A 用のセンス・データは保持され続ける．イニシエータ A がセンス・データを読み出すと CA 状態は解除される

たは他のイニシエータから発行された複数個のコマンドを受け付けてキューイングします．

仕様上はイニシエーター-ロジカル・ユニットごとに最大 256 個までのコマンドのキューイングが可能ですが，実際にキューイング可能なコマンドの数はイニシエータおよびターゲットの組み込み仕様に依存します．

イニシエータはコマンド発行時に IDENTIFY メッセージに続けていずれかのキュー・タグ・メッセージを発行し，各コマンドに固有な識別子(キュー・タグ)を与えます．イニシエータはその時点でそのロジカル・ユニット上で実行中またはキューイング中のコマンド(そのイニシエータが発行したコマンド)のキュー・タグと重複しない値を指定します．

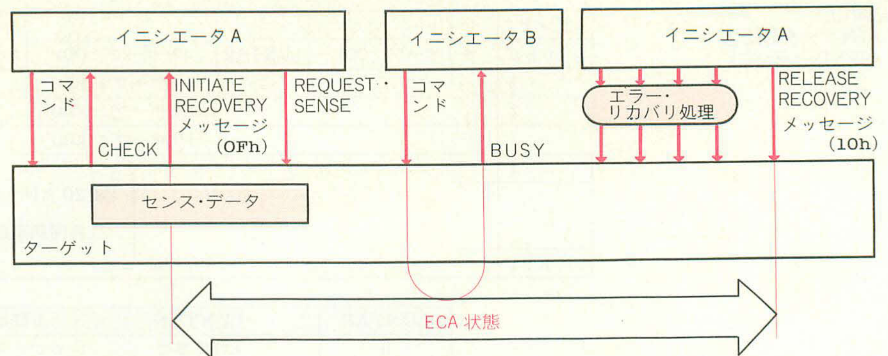
イニシエータ内の SCSI ポインタ機構ではキューイングさせるコマンド数分のセーブ・ポインタを用意し，ターゲット ID/LUN/キュー・タグの値に基づいてポインタのセーブやリストアを行います．

ターゲットはコマンド・キューをロジカル・ユニット単位に管理し，コマンド・キューに空きがなくなったときは新たなタグ付きコマンドに対してキュー・フル・ステータスを応答します．また，タグ付きコマンドのリコネクションを実行するときにはターゲットはリコネクション・シーケンスで IDENTIFY メッセージに続けて SIMPLE QUEUE TAG メッセージを発行し，リコネクションが必要なコマンドのキュー・タグをイニシエータに通知します．

図 19 にタグ付キューイングの実行例を示します．SIMPLE QUEUE TAG メッセージで発行されたコマンドはターゲットがその実行順序を変更できます．ハードディスク装置では一般にリード/ライト・ヘッドの移動が最小になるようにコマンドを並べ変えます(このためシーク・リオーダリングと呼ばれることがある)．ORDERED QUEUE TAG メッセージで発行されたコマンドは必ず到着順に，また，HEAD OF QUEUE TAG メッセージで発行されたコマンドは実行中のコマンドの終了後に最優先で実行されます．

コマンドがエラー終了した場合，そのロジカル・ユニットにキューイングされている後続のコマンドをターゲットがどう扱うかについてはつぎのふたつの方法が規定されており，モード・セレクト・パラメータで指

〈図18〉ECA状態



ターゲットがINITIATE RECOVERY・メッセージを送信するとイニシエータAに対するECA状態が発生する．ECA状態の時に他のイニシエータ(図では，イニシエータB)からコマンドを受け取った時はターゲットはBUSYステータスを応答する．イニシエータAは一連のエラー・リカバリ処理の間，そのターゲットとロジカル・ユニットを占有できる．イニシエータAがRELEASE RECOVERYメッセージを発行するとECA状態は解除される

定することができます．

① キューイングされているコマンドはそのまま保持し，イニシエータにその処置を委ねる

② CAまたはECA状態が解除されたとき，キューイングされているすべてのコマンドをクリア

どちらの場合もCAまたはECA状態の間はターゲットはキューイングされている後続のコマンドの実行は開始しません．また，イニシエータはエラー・リカバリ処理のためのコマンドはタグなしで実行します．

①の方法ではECA状態が生成されていればイニシエータはコマンド・キュー内のコマンドを選択的にクリアすることなどができます．SCSI2ではエラー・リカバリ系や例外条件処理の細部までは規定されていませんが，今後タグ付きキューイング機能が一般化する過程で，徐々に標準形が確立されていくと思われます．

なお，タグ付きキュー・イングをサポートしていないターゲットがキュー・タグ・メッセージを受け取ったときは，ターゲットはそのメッセージをリジェクトして処理を継続し，タグなしコマンドとして実行します．

● **オーバ・ラップ・コマンド**

コマンドの実行が終了する前に同一のイニシエータが同一のロジカル・ユニットに対してつぎのコマンドを発行(タグ付きキューイングのときは同じキュー・タグのコマンド)したとき，またはひとつのイニシエータがタグ付きコマンドとタグなしコマンドを混在して発行しようとしたとき(タグ付きキューイングでのCAまたはECA状態の場合を除く)はオーバラップ・コマンドになります．オーバラップ・コマンドが発生したときターゲットは，そのイニシエータがそのロジカル・ユニットに発行していたすべてのコマンド(実行中あるいはキューイング中)をアボートします．

共通コマンドの仕様

すべてのデバイス・タイプに共通なコマンドの一覧を**表11**に示します．SCSI2ではTEST UNIT READY，REQUEST SENSE，INQUIRY，SEND DIAGNOSTICの4つのコマンドのサポートが必須になっています．

追加コマンドおよび機能が拡張された主なコマンドの概要を以下に説明します．

● **REQUEST SENSE：03h**

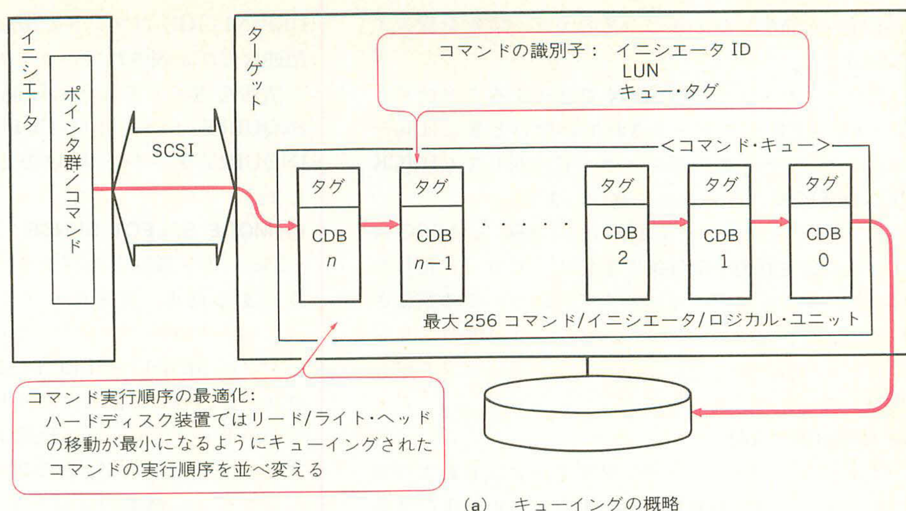
CAやECA状態が発生したとき，イニシエータは直ちにこのコマンドを発行してエラー・リカバリのための情報(センス・データ)を読み出します．

SCSI1およびCCSではこのコマンドのCDBで転送バイト長(Allocation Length)にゼロが指定されたとき，ターゲットはセンス・データの先頭4バイトを転送することになっていましたが，SCSI2の仕様ではターゲットは何も転送せずにコマンドを終了させ，センス・データをクリアします．

センス・データは**図20**に示す形式(従来，Extendedセンスと呼ばれていた形式)が標準になり，SCSI1で許容されていた4バイト長のセンス・データ形式は削除されました．ターゲットは少なくとも18バイト長のセンス・データをサポートする必要があります．エラー・コード：70hのセンス・データは従来と同様にCAまたはECA状態が発生したときのコマンドに関するエラー情報を示します．

一方，エラー・コード：71hのセンス・データはデファード・エラーと呼ばれ，すでにイニシエータにGOODスターテスを報告したコマンド(イミディエート・ビットの指定で実動作の完了前に正常終了を報告したときやライト・バック・キャッシュなど)の実行中

〈図 19〉
タグ付きコマンド・キューイング



(a) キューイングの概略

キュー・タグ：02～08 のコマンド ⇨

論理ブロック・アドレス：10000 から 100 ブロックを読み出すリード・コマンドを実行中に七つのコマンドがキューイングされた（この動作例ではコマンド・キューの深さは 8 とする）

コマンド・キュー

	キュー・タグ	コマンド	先頭論理ブロック・アドレス	ブロック数	ステータス
0	SQ:01	READ	10000	100	⇦ 実行中
1	SQ:02	READ	100	2	
2	SQ:03	READ	11000	100	
3	SQ:04	READ	12000	100	
4	OQ:05	WRITE	200	2	
5	SQ:06	READ	1000	100	
6	SQ:07	READ	210	2	
7	SQ:08	WRITE	10000	100	

実行中のリード・コマンドが終了する. キューの中のコマンドの実行順は次のように並べ換えられる
・並べ替えの対象は SIMPLE QUEUE TAG のみ
・ORDERED QUEUE TAG は飛び越さない

コマンド・キュー

	キュー・タグ	コマンド	先頭論理ブロック・アドレス	ブロック数	ステータス
0	SQ:03	READ	11000	100	⇦ 次に実行
1	SQ:04	READ	12000	100	
2	SQ:02	READ	100	2	
3	OQ:05	WRITE	200	2	
4	SQ:07	READ	210	2	
5	SQ:06	READ	1000	100	
6	SQ:08	WRITE	10000	100	
7					

HEAD OF QUEUE TAG のコマンド ⇨

HEAD OF QUEUE TAG で発行されたコマンドは現在実行中のコマンドが終了したら，その次に実行されるようにキューイングする

コマンド・キュー

	キュー・タグ	コマンド	先頭論理ブロック・アドレス	ブロック数	ステータス
0	SQ:03	READ	11000	100	⇦ 実行中
1	HQ:01	READ	50	1	
2	SQ:02	READ	100	2	
3	SQ:04	READ	12000	100	
4	OQ:05	WRITE	200	2	
5	SQ:07	READ	210	2	
6	SQ:06	READ	1000	100	
7	SQ:08	WRITE	10000	100	

新たなコマンド ⇨
QUEUE FULL ステータス ⇦

コマンド・キューに空きがない時に新たなコマンドを受け取った場合は QUEUE FULL ステータスを応答する

SQ:SIMPLE QUEUE TAG
OQ:ORDERED QUEUE TAG
HQ:HEAD OF QUEUE TAG

(b) キューイングの例

にエラーが発生したときに，そのエラー情報を報告するものです．

デファード・エラーはAENで通知することができますが，AENがサポートされていないときには同一イニシエータからのつぎのコマンドに対してCHECK CONDITIONステータスを通知します．

センス・データのバイト12～13のASCとASCQはエラーの発生原因や例外条件を分類してコード化したもので，すべてのデバイス・タイプについて体系化されています．イニシエータはほとんどの場合，センス・キー/ASC/ASCQによりエラー・リカバリ手順を選択することができます．

● INQUIRY：12h

イニシエータはこのコマンドでターゲットおよびロジカル・ユニットの属性やSCSIのレベルおよびサポートされている機能などを知ることができます．

▶スタンダードINQUIRYデータ

このデータはデバイス・タイプなどの装置の一般属性を報告します．図21に示すようにCCSの規定と同様のデータ形式になっていますが，オプション機能のサポート・レベルを表示するフィールドが追加されています．

ターゲットは少なくともバイト0～35までの36バイト長のINQUIRYデータをサポートする必要があります．

▶ VPD(Vital Product Data)

ターゲットやロジカル・ユニットのベンダ固有情報を報告する機能としてVPDが追加(オプション)されました．VPDには装置シリアル番号，センス・データで報告されるFRUコードの説明の記述，CHANGE DEFINITIONコマンドで指定が可能な動作モードの記述などが定義されていますが，その他のベンダ固有の情報を追加することも可能です．イニシエータはINQUIRYコマンドのCDB上で，スタンダードINQUIRYデータとVPDのどちらを要求するか指定します．

● MODE SELECT/SENSE：15/55/1A/5Ah

256バイト長以上のパラメータ・リストを扱えるようにするため，従来の6バイト長CDBに加えて10バイト長CDB(グループ2)が追加されました．グループ2のMODE SELECT/SENSEコマンドではパラメータのヘッダ部も8バイト長に拡張されています．

モード・パラメータの形式はSCSI1ではベンダ固有になっていましたが，SCSI2ではCCSで規定されたページ形式が標準形になりました．また，新機能(統計情報，ECA，AEN，タグ付きキューイング)をコントロールするためのパラメータ・ページ(ページA：Control Mode Page)などが追加になったほか，CCSと同一のパラメータ・ページでもページ長を拡張して機能を追加したものがあります．

モード・パラメータはターゲットよってサポート範囲が違ったり，将来の規格で機能が追加される可能性があります．イニシエータはMODE SENSEコマンドを使用してターゲットがサポートしているパラメータの種類や長さとその属性を調べ，それに基づいてMODE SELECTコマンドを発行するようにし，特定の仕様レベルに依存しないようにしておくべきです．

● SEND/RECEIVE DIAG.：1D/1Ch

SEND DIAG.コマンドのセルフ・テスト機能のサポートは必須条件になっています．イニシエータはこの

〈表11〉
全デバイス・タイプ共通のコマンド・セット

区分	OPコード	コマンド名称	SCSI2	CCS	SCSI1
グループ0	00	TEST UNIT READY	M	M	O
	03	REQUEST SENSE	M	M	M
	12	INQUIRY	M	M	E
	15	MODE SELECT (6)	Z	O	O
	18	COPY	O	O	O
	1A	MODE SENSE (6)	Z	O	O
	1C	RECEIVE DIAGNOSTIC RESULTS	O	O	O
	1D	SEND DIAGNOSTIC	M	M	O
グループ1	39	COMPARE	O	O	O
	3A	COPY AND VERIFY	O	O	O
	3B	WRITE BUFFER	O	O	RSVD
	3C	READ BUFFER	O	O	RSVD
グループ2	40	CHANGE DEFINITION	O	RSVD	RSVD
	4C	LOG SELECT	O	RSVD	RSVD
	4D	LTG SENSE	O	RSVD	RSVD
	55	MODE SELECT (10)	Z	RSVD	RSVD
	5A	MODE SENSE (10)	Z	RSVD	RSVD

M：サポート必須，E：拡張仕様，O：オプション，
Z：デバイス・タイプ依存，RSVD：リザーブ

機能を使用してターゲットに自己診断を実行させることができます．自己診断の結果はステータスで通知されるのでセルフ・テスト機能だけを使用するときはRECEIVE DIAG. RESULTSは必要としません．

オプション機能としてSEND DIAG.コマンドでパラメータを送ってターゲットに特定の診断動作などを実行させ，その結果をRECEIVE DIAG. RESULTSコマンドで取り出すこともできます．このときのパラメータにもページ形式やターゲットのサポート範囲を調べる機能が導入され，装置個別仕様からの分離が図られています．

ダイレクト・アクセス・デバイスではこのオプション機能を使用してデータ・ブロックのアドレス変換(論理アドレス→物理アドレス)を行うための標準パラメータが規定されています．

● WRITE/READ BUFFER：3B/3Ch

これらはCCSで追加されたコマンドですが，64Kバイト長以上のデータ・バッファを操作しやすいよう機能が拡張されました．

また，READ BUFFERコマンドでターゲットのデータ・バッファの構成を調べる機能が追加され，装置個別仕様からの独立性が高められています．さらに，WRITE BUFFERコマンドにはマイクロコードのダウン・ロード機能(イニシエータからターゲットのマイクロコードや制御情報を転送する機能)が追加され，オンラインでのバージョン・アップなどに対応できるようになっています．

● CHANGE DEFINITION：40h

プログラマブル動作モードの項で述べたようにイニシエータはこのコマンドでターゲットとロジカル・ユニットの動作モードを，SCSI2，SCSI1，CCSのいずれか，あるいはベンダ固有の特殊動作モードに変更することを指定できます．

動作モードの変更適用範囲(指定されたロジカル・ユニットのみ/ターゲット全体，このコマンドを発行したイニシエータに対してのみ/すべてのイニシエータに対して)はターゲットの組み込み仕様に依存します．動作モードの変更が他のイニシエータからのコマンドの動作に影響を与えるときは，ターゲットは，それらのイニシエータに対してユニット・アテンション・コンディションを生成し，動作モードが変更されたことを通知します．

● LOG SELECT/SENSE：4C/4Dh

これらのコマンドはターゲットが累積する統計情報を操作するものです．統計情報にはSCSIバスの動作状態に関連したバッファ・オーバラン/アンダランの発生回数や，I/O装置のメカニカル動作および媒体でのエラー発生回数などがあります．イニシエータはLOG SELECTコマンドで，個々の統計情報の初期化やしきい値を設定したり，累積の開始や停止を指示し，LOG SENSEコマンドで累積値を読み出すことができます．ターゲットは統計情報の対象となる事象の発生回数を累積し，しきい値または最大値に達するとユニット・アテンション・コンディションを生成することができます．

統計情報はSCSIバスの動作や負荷バランスをチュ

〈図20〉
センス・データの標準形式

ビット	7	6	5	4	3	2	1	0
バイト0	Valid	エラーコード：70hまたは71h						
バイト1	セグメント番号(COPY，COMPARE，COPY AND VERIFY)							
バイト2	Filemark	EOM	ILI	リザーブ=0	センス・キー			
バイト3	[MSB] インフォメーション							
バイト4								
バイト5								
バイト6	[LSB]							
バイト7	追加センス・データ長($n-7$)							
バイト8	[MSB] コマンド固有インフォメーション (Command-Specific Information)							
バイト9								
バイト10								
バイト11	[LSB]							
バイト12	ASC(Additional Sense Code)							
バイト13	ASCQ(Additional Sense Code Qualifier)							
バイト14	FRUコード(Field Replaceable Unit Code)							
バイト15	SKSV	センス・キー固有インフォメーション						
バイト16								
バイト17								
バイト18	追加センス・バイト (コマンド固有，デバイス固有，またはベンダ固有)							
≈								
バイトn								

〈図21〉スタンダードINQUIRYデータ

ビット	7	6	5	4	3	2	1	0
バイト 0	Peripheral Qualifier			デバイス・タイプ・コード				
バイト 1	RMB	デバイス・タイプ修飾子						
バイト 2	ISO Version		ECMA Version			ANSI Version		
バイト 3	AENC	TrmIOP	(リザーブ：00)		Response Data Format			
バイト 4	追加データ長（n −4）							
バイト 5	(リザーブ：00h)							
バイト 6	(リザーブ：00h)							
バイト 7	RelAdr	WBus32	WBus16	Syns	Linked	リザーブ:0	CmdQue	SftRe
8 〜 15	ベンダID〔ASCII〕							
16 〜 31	プロダクトID〔ASCII〕							
32 〜 36	プロダクト版数〔ASCII〕							
36 〜 55	ベンダ固有							
56 〜 95	(リザーブ :00h)							
96 〜 n	ベンダ固有							

ANSI Version：
000：未適合
001：SCSI1
010：SCSI2

Response Data Format：
0000：SCSI1
0001：CCS
0010：SCSI2

オプション機能のサポート・レベルを表示するビット（1：サポート，0：未サポート）
AENC：ATN受信機能（プロセッサ・デバイスのみ）
TrmIOP：TERMINATE I/O PROCESS メッセージ
RelAdr ：相対アドレス機能（CDBの“RelAdr”ビット）
WBus32：32ビット幅のWIDE転送
WBus16：16ビット幅のWIDE転送
Sync：同期モードのデータ転送
Linked：コマンド・リンク機能（CDBの“Link”ビット）
CmdQue：タグ付きコマンド・キューイング
SftRe：ソフト・リセット・コンディション

ーニングするための情報採取やI/O装置の予防・保守のための情報採取などに効果的な機能です．

ダイレクト・アクセス・デバイスのコマンド

ダイレクト・アクセス・デバイス用のコマンドの一覧を表12に示します．ダイレクト・アクセス・デバイスについてはCCSで標準のコマンド・セットが確立されていたため，大きな変更はありませんが，

① キャッシュ制御機能の追加
② 診断系コマンドの強化
③ ディスク装置の回転同期（スピンドル・シンク）制御機能の追加

などが行われています．

● **キャッシュ制御用のコマンド**

キャッシュ・メモリを備えた装置の動作を制御するために次の3つのコマンドが用意されました．

▶ PRE-FETCH：34h

このコマンドはCDBで指定された範囲のデータを媒体からキャッシュ・メモリ上に読み込みます（イニシエータには転送しない）．

ターゲットはキャッシュ上のロックされていない領域にデータを読み込み，指定されたすべてのデータが正常に読み込めたときにはCONDITION METステータスを報告します．

▶ SYNCHRONIZE CACHE：35h

このコマンドはCDBで指定された範囲のデータ・ブロックについてキャッシュ・メモリ上の値と媒体上の値を一致させるものです．

ライト・バック・キャッシュでまだ媒体上に書き込まれていないデータ・ブロック（このコマンドのCDBでの指定範囲内で）があればターゲットはそのデータを媒体上に書き出します．

▶ LOCK-UNLOCK CACHE：36h

このコマンドはキャッシュ・メモリ内のデータに対してターゲットのキャッシュ・アルゴリズムによる追い出しを禁止（ロック）または許可（アンロック）します．

データの範囲はCDB上の論理ブロック・アドレスとブロック数により指定しますが，このコマンドが発

行された時点でキャッシュ上に存在しているデータ・ブロックだけがロック/アンロックの対象になります．

ロックされたデータはこのコマンドでアンロックが指定されるまでキャッシュ上に保持されます．

● キャッシュ制御用モード・セレクト・パラメータ

キャッシュ・メモリを備えた装置は，通常，READコマンドで指定されたデータ・ブロックを媒体から読み出した後，後続のデータ・ブロック(イニシエータからは要求されていない)をキャッシュ上に先読みしますが，先読みの範囲や先読み動作とつぎのコマンド実行との優先度などを制御するためのモード・セレクト・パラメータ（ページ8：Caching Page，図22)が追加されました．イニシエータはこのパラメータを使用してWRITEコマンドとREADコマンドのキャッシュ動作の禁止/許可を行うこともできます．

● CDB上のキャッシュ制御用フラグ

コマンドの発行単位でキャッシュの動作を制御できるようにするため以下のコマンドのCDB上に2種類の制御フラグ(DPO，FUA)が追加されました．

・READ(10)	：DPO，FUA
・WRITE(10)	：DPO，FUA
・WRITE AND VERIFY	：DPO
・VERIFY	：DPO

▶ DPO(CDBバイト1，ビット4)

DPO(Disable Page Out)フラグはそのコマンドの実行によってすでにキャッシュ上にある他のデータを置き換えてよいかどうかを指定します．DPOが1のときにはそのコマンドでアクセスされたデータ・ブロックは最低位の優先順位でキャッシュに保持すればよいということを指定します．つまり，キャッシュ上に存在しているデータの追い出しは禁止されます．

そしてDPOが0のときにはそのコマンドの実行時にキャッシュ・データの置き換えを行ってもよいことを示します．

▶ FUA(CDBバイト1，ビット3)

FUA(Force Unit Access)フラグはそのコマンドを実行するときに媒体アクセスを強制するかどうかを指定します．

FUAが0のときにはWRITEコマンドはキャッシュ上にデータを格納しただけで終了(ライト・バック・キャッシュ)し，READコマンドは指定されたデータがキャッシュにヒットしていれば媒体をアクセスすることなくキャッシュからデータを転送します．

一方FUAが1のときにはそのコマンドは必ず媒体をアクセスして実行することを指定します．WRITEコマンドは媒体上へのデータの書き込みが完了してから終了ステータスを報告します(ライト・スルー・キャッシュ)．READコマンドはキャッシュにヒットしていても必ず媒体からデータを読み出します．

なお，VERIFY系のコマンドにはFUAフラグはありませんが，いつもFUA=1であるものとして，つまり必ず媒体上のデータに対してベリファイ動作が行われます．ここで，FUA=1のREADコマンドまたはVERIFY系のコマンドではコマンドの実行前に暗黙的にSYNCRONIZE CACHEコマンドと同一の動作が実行されます．つまり対象データがまだ媒体に書き込まれていない場合にはその書き込みを実行してからベリファイまたはリードを行います．

● READ LONG/WRITE LONG：3E/3Fh

これらのコマンドは媒体上のデータ・ブロックのデータ部とそのエラー訂正コード(ECC)を直接リード/ライトします．

イニシエータはエラー・リカバリ処理のテストなどのためにREAD LONGコマンドで読み出したデータの一部を変更してWRITE LONGコマンドで書き戻すことにより，訂正可能または訂正不可能なエラーが発生するデータ・ブロックを媒体上に任意に作成することができます．

● WRITE SAME：41h

このコマンドはイニシエータから転送された1ブロック分のデータを媒体上の指定された領域に繰り返してライトします．

このとき，各データ・ブロックの先頭にそのブロックの論理アドレスまたは物理アドレスを書き込むこともできます．この機能はテストなどの目的で媒体を一定のデータ・パターンで初期化するときなどに便利です．

● 回転同期(スピンドル・シンク)制御機能

ディスク・アレイなどのアプリケーションでは複数台のディスク装置のスピンドル・モータの回転を同期させることがありますが，イニシエータからその制御を行うための機能としてモード・セレクト・パラメータのページ4(Rigid Disk Drive Geometry Page)が拡張されました．

モード・セレクト・コマンドによってスピンドル・シンクのモード(禁止，スレーブ，マスタ，マスタ・コントロール)や回転同期のオフセット(基準信号への同期角度)を指定することができます．また，回転同期の完了や同期異常はユニット・アテンション・コンディションで通知されます．

● その他の拡張機能

▶モード・セレクト・パラメータ：エラー・リカバリ用パラメータはリード/ライト用のパラメータ(ページ1：Read-Write Error Recovery Page)とベリファイ用のパラメータ(ページ7：Verify Error Recovery Page)に分離されました．

また，CDR〔Constant Density Recording：ZBR(Zone Bit Recording)とも呼ばれ，内周と外周シリンダで記憶密度(BPI)を均一化することで外周シリン

〈表12〉 ダイレクト・アクセス・デバイスのコマンド・セット

＊：全デバイス・タイプ共通のコマンド

区分	OPコード	コマンド名称	キャッシュ・フラグ	サポートの要否 SCSI2	CCS	SCSI1
	00*	TEST UNIT READY		M	M	O
	01	REZERO UNIT		O	O	O
	02			V	V	V
	03*	REQUEST SENSE		M	M	M
	04	FORMAT UNIT		M	M	M
	05			V	V	V
	06			V	V	V
	07	REASSIGN BLOCKS		O	O	O
	08	READ (6)		M	M	M
	09			V	V	V
	0A	WRITE (6)		M	M	M
	0B	SEEK (6)		O	O	O
グループ 0	0C～11			V	V	V
	12*	INQUIRY		M	M	E
	13			V	V	V
	14			V	V	V
	15*	MODE SELECT (6)		O	O	O
	16	RESERVE		M	M	O
	17	RELEASE		M	M	O
	18*	COPY		O	O	O
	19			V	V	V
	1A*	MODE SENSE (6)		O	O	O
	1B	START/STOP UNIT		O	O	O
	1C*	RECEIVE DIAGNOSTIC RESULTS		O	O	O
	1D*	SEND DIAGNOSTIC		M	M	O
	1E	PREVENT-ALLOW MEDIUM REMOVAL		O	O	O
	1F			RSVD	RSVD	RSVD
	20～24			V	V	V
	25	READ CAPACITY		M	M	E
	26			V	V	V
	27			V	V	V
	28	READ (10)	DPO FUA	M	M	E
	29			V	V	V
	2A	WRITE (10)	DPO FUA	M	M	E
	2B	SEEK (10)		O	O	O
	2C			V	V	V
	2D			V	V	V
	2E	WRITE AND VERIFY	DPO	O	O	O
	2F	VERIFY	DPO	O	O	O
	30	SEARCH DATA HIGH		O	O	O
グループ 1	31	SEARCH DATA EQUAL		O	O	O
	32	SEARCH DATA LOW		O	O	O
	33	SET LIMITS		O	O	O
	34	PRE-FETCH		O	RSVD	RSVD
	35	SYNCRONIZE CACHE		O	RSVD	RSVD
	36	LOCK/UNLOCK CACHE		O	RSVD	RSVD
	37	READ DEFECT DATA		O	O	RSVD
	38			RSVD	RSVD	RSVD
	39*	COMPARE		O	O	O
	3A*	COPY AND VERIFY		O	O	O
	3B*	WRITE BUFFER		O	O	RSVD
	3C*	READ BUFFER		O	O	RSVD
	3D			RSVD	RSVD	RSVD
	3E	READ LONG		O	RSVD	RSVD
	3F	WRITE LONG		O	RSVD	RSVD
	40*	CHANGE DEFINITION		O	RSVD	RSVD
	41	WRITE SAME		O	RSVD	RSVD
	42～4B			RSVD	RSVD	RSVD
	4C*	LOG SELECT		O	RSVD	RSVD
グループ 2	4D*	LOG SENSE		O	RSVD	RSVD
	4E～54			RSVD	RSVD	RSVD
	55*	MODE SELECT (10)		O	RSVD	RSVD
	56～59			RSVD	RSVD	RSVD
	5A*	MODE SENSE (10)		O	RSVD	RSVD
	5B～5F			RSVD	RSVD	RSVD
グループ3, 4, 5	60～BF			RSVD	RSVD	RSVD
グループ6, 7	C0～FF			V	V	V

M：サポート必須，E：拡張仕様，O：オプション，RSVD：リザーブ
V：ベンダ固有

〈図 22〉キャッシュ制御用のモード・セレクト・パラメータ

ビット	7	6	5	4	3	2	1	0
バイト 0	PS	リザーブ:0	ページ・コード:08h					
バイト 1	ページ長:0Ah							
バイト 2	(リザーブ:00000)					WCE	MF	RCD
バイト 3	リード・データの保持優先度 (Demand Read Retention Priority)				ライト・データの保持優先度 (Write Retention Priority)			
バイト 4	プリフェッチ抑止ブロック数 (Disable Pre-fetch Transfer Length)							
バイト 5								
バイト 6	最小プリフェッチ (Minimum Pre-fetch)							
バイト 7								
バイト 8	最大プリフェッチ (Maximum Pre-fetch)							
バイト 9								
バイト 10	最大プリフェッチ制限ブロック数 (Maximum Pre-fetch Ceiling)							
バイト 11								

● WCE
0：ライト・スルー
1：ライト・バック
● RCD
0：キャッシュ許可
1：キャッシュ禁止

● リード・データの保持優先度
0: 他のデータと同列
1: 低優先度
F: 高優先度

● ライト・データの保持優先度
0: 他のデータと同列
1: 低優先度
F: 高優先度

● READ コマンド終了後のキャッシュへの先読み動作
・先読みの許可：(RCD=0)＆(READ コマンドのブロック数≦プリフェッチ抑止ブロック数)
・最小プリフェッチ量：・If MF=0, =(最小プリフェッチ)
・If MF=1, =(最小プリフェッチ)＊(READ コマンドのブロック数)
・最小プリフェッチ量分の先読み動作は，次のコマンドより優先
・最大プリフェッチ量：・If MF=0, =(最大プリフェッチ)
：・If MF=1, =(最大プリフェッチ)＊(READ コマンドのブロック数)
・ただし，上限は(最大プリフェッチ制限ブロック数)

キャッシュ機能をさらに高度化するため ANSI ではこのページの機能拡張(セグメント・キャッシュなど)を検討している

ダのトラック容量を増加させ，大容量化を図る手法〕に関連したパラメータの追加や，フレキシブル・ディスク用のパラメータの機能追加も行われています．CCS で規定されたパラメータ・ページの内，ページ 1，2，4，5 などは機能追加によりページ長が変更されているので注意が必要です．

▶ FORMAT UNIT：フォーマッティングのときに媒体に書き込む“初期化データ・パターン”を指定する機能やイミディエート機能(CDB とパラメータの転送が終わった時点で終了ステータスを報告する)などが追加されました．

▶ RESERVE/RELEASE：異なったイニシエータからのリザーブ要求をキューイングする機能は削除されました．また，サード・パーティ・リザーブのときに，RESERVE コマンドを発行したイニシエータのモード・パラメータをサード・パーティ・デバイス用のモード・パラメータにコピーする機能が設けられました．

＊　　＊

1990 年の中頃から FAST SCSI やキュー・タグ・メッセージを組み込んだプロトコル IC が登場しはじめており，最近では WIDE SCSI への対応も図られてきているので本格的な SCSI2 仕様の装置が続々と登場してくることは間違いありません．

しかし，タグ付きコマンド・キューイングなどの新しい機能はソフトウェアの対応も含めて徐々にシステムへの組み込みが進んでいくと思われるので，実使用環境に適した標準形が確立され一般化するまでにはもう少し時間がかかると予想されます．

ANSI ではすでに，SCSI2 の機能をさらに発展させた SCSI3 の検討が開始されています．今後ますます進展が予想されるなかで SCSI2 への取り組みは必要不可欠になっています．

◆参考・引用*文献◆

(1)*ANSI, X3T9.2/86 - 109 Revision 10c, X3T9/89 - 042, draft proposed American National Standard for Information Systems - Small Computer System Interface - 2 (SCSI - 2), American National Standard Committee X3, March 9, 1990.

(2) ANSI, X3.131 - 1986, American National Standard for Information Systems - Small Computer System Interface (SCSI), American National Standard Intitiute, 1986.

(3) ANSI, X3T9.2/85 - 52, Common Command Set (CCS) of the Small Computer System Interface (SCSI), REV 4.B. American National Standard Committee X3, 1986.

(4)*菅谷誠一；「SCSI 規格の詳細解説」，最新 SCSI マニュアル，CQ 出版株式会社.

編集雑記

編集部から

● ハードディスクは本文中でも解説されているように非常に高度な技術を使っています．

そのようなわけで実際，ハードディスク・ユニットそのものを作ることやユニットの中身を改造することなどは個人の作業がおよぶ領域ではありません．もはや入出力がはっきりとわかっているブラックボックスとしてあつかうしかありません．

それと同じく製品としてのハードディスクの見た目もただの箱です．しかも外付け型ならば箱として見えるものの内蔵型ともなるとあるのかないのか全く判別がつきません．

このようにハードディスクにはある種の得体の知れなさがあります．少なくともこの文章を書いている本人はそう思っています．

こういった得体の知れなさがアマチュア製作への取り付きづらさや，そもそもハードディスクを個人でいじくり回してみようという気にさせてくれない原因になっているのではないかとも考えます．

しかし，ハードディスク・ユニットの中身に手を加えることが不可能だとしても，取り付くしまが全くないわけではありません．ユニットの外側にならいくらでも手を加える余地が残っています．

● そのような理由で今回の特集ではハードディスク自身についての解説は第一章から第三章で一通り行なうだけにしました．他の章はすべてハードディスクをつなぐインターフェースであるSCSIの解説と使い方です．

第五章および第六章は現在個人で扱えるハードディスク関連の題材としてはかなりいい線をいっているような気がします．

もっともハードウェアの最近の傾向である1チップ化の波はここでも例外ではなく，ハードウェアは機能ICのつなぎ合わせで残りはすべてソフトウェアという手法を取るのは仕方のないことです．

● 今回のページの都合上，具体的な製作例を紹介することができませんでした．そういった事例に関しては本誌「トランジスタ技術」誌上でおいおい御紹介できると思います．

また，第六章として掲載した内容はもともとNIFTY-SERVEのFDEVICEという電子会議をはじめとしたパソコンネット上で企画，検討，製作されたものであることをここでおことわりしておきます．興味を持たれた方はこちらをのぞいてみるのもいかがでしょうか．第六章で紹介したソフト&ハードはもちろんのこと，その他にも興味深い内容が数多くアップされています．

● SCSI2によってWIDEとFASTの二種類の拡張がなされたSCSIは，これまで以上に使用範囲が広がるはずです．ハードディスクの他，考えられるだけでもMO，MT，DAT，CD-ROM，WORM，プリンタ，プロッタ，イメージ・スキャナ，他のマイコン・システムなどさまざまです．マイコンの大容量化，高速化がなされていく以上，高速インターフェースの需要が無くなるわけがありません．

今，例に挙げたもののうち，いくつかはまだ製品として市場に出回っていません．それらは次の主流としてひかえているものばかりです．いずれ，それらが市場に出回るようになる頃，SCSIは現在のセントロニクス・バスやRS 232 Cのように誰もが簡単に製作できる入門に最適な簡単明瞭インターフェースと言われるようになるでしょうか？　これが編集を終えた後，私がいだいた，ささやかな疑問です．　(☢)

次号のお知らせ（6月28日発売）

特集　新・電源回路設計のすべて

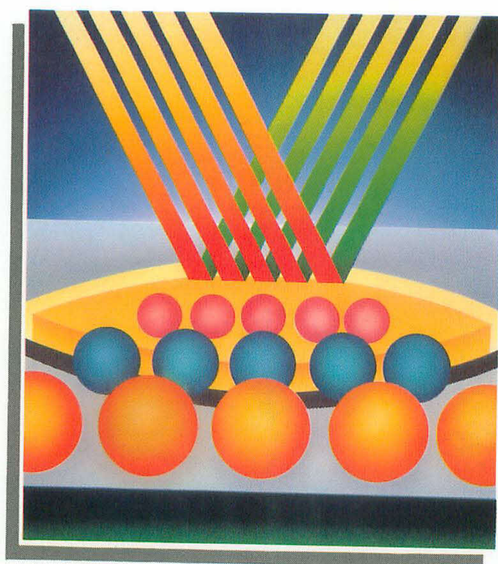
あらゆる電子装置に必要な電源回路技術を，実際に設計・製作するプロセスを通じて詳解します．

3端子レギュレータ回路や低損失型のリニア電源，RCC方式やフォーワード・コンバータ方式などのSW電源，また近年注目されている共振型SW電源について詳しく説明します．また，電源用部品や新しい安全規格の解説も行う予定です．

トランジスタ技術 SPECIAL　No. 27

発行所　CQ出版株式会社　Printed in Japan
〒170 東京都豊島区巣鴨 1-14-2
電　話　編集部：03(5395)2123，広告部：03(5395)2132
営業部：03(5395)2141
振　替　東京 0-10665
発行人　神戸一夫
編集人　蒲生良治

1991年5月1日発行　印刷・製本　三晃印刷株式会社



CQ出版社 定価1,540円(本体1,495円) ISBN4-7898-3199-X C3055 P1540E